Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un diseñador de informes y listados en el marco de un ERP

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Alejandro Lozano Jiménez

Tutor: Patricio Letelier Torres

Curso 2020-2021

Resumen

Los *Enterprise Resource Planning (ERP)* son herramientas de gran utilidad en el contexto empresarial, dado que permiten a una organización gestionar sus procesos de negocio e información mediante la integración de diferentes aplicaciones, así como automatizar la planificación, gestión y trazabilidad de los recursos. Una de esas aplicaciones o módulos de gran importancia en estos *ERPs*, sería el de Listados e Informes, que permite extraer información del sistema y presentarla unificadamente para apoyar los procesos de toma de decisiones. Para el diseño de las Plantillas asociadas a estos Listados e Informes, existen los conocidos como Diseñadores de Plantillas. Estos Diseñadores, buscan ofrecer herramientas al diseño de estas Plantillas, de forma que se establezcan partes fijas, comunes para todas las instancias o generaciones de un Listado o Informe dado, y otras variables, que dependerán de los datos de la instancia o generación concreta de dicho Informe o Listado.

En el presente documento se expondrán las diferentes mejoras llevadas a cabo sobre un Diseñador de Plantillas en formato *WordprocessingML* de Listados e Informes, con el objetivo que dichas mejoras sean suficientes para incrementar las funcionalidades y posibilidades de diseño que el Diseñador ya ofrece. Durante el desarrollo del proyecto, se expondrán las diferentes especificaciones de requisitos, decisiones de diseño, problemas encontrados y resultados obtenidos durante y tras la implementación de dichas mejoras.

Palabras clave: Informes, Listados, Plantillas, Diseñador de Plantillas, *Enterprise Resource Planning (ERP)*.

Abstract

Enterprise Resource Planning (ERP) are tools that offer great value for businesses, since they allow any organization to manage their business processes and information by the integration of several applications, as well as enabling for the automation of the planning, management, and traceability of resources. One of those applications or modules of great value, is the *Report* module, which allows for extracting information from the system and presenting it in a way that assists the decision-making process. For designing those Templates associated to Reports, there are the so-called Templates Designers. Those Designers present several features and tools for assisting in the design of those Templates, in a way that there are some fixed parts, common to all the instances or generations of a given Report, and other variable parts, which depend on the data of the instance or generation of this given Reports.

In this document we will present several enhancements to a Reports Templates Designer for the WordprocessingML format, so that those enhancements improve the functionality and design possibilities of that same Designer. Through the development of this project, the different requirement specification, design decisions, encountered difficulties and results acquired during and after the implementation of those enhancements will be explained.

Keywords: Reports, Templates, Templates Designer, *Enterprise Resource Planning (ERP)*.

Índice

Tabla de figuras

Tabla de acrónimos

1. Introducción

A continuación, se explicarán los conceptos más relevantes en *reporting*, para continuar con la presentación de los objetivos del trabajo, y terminar hablando de la estructura del resto del trabajo.

1.1 Motivación y contexto

En el contexto de la gestión empresarial, los *Enterprise Resource Planning (ERP)* son herramientas de gran utilidad dado que permiten a una organización gestionar sus procesos de negocio e información mediante la integración de diferentes aplicaciones, de forma que facilitan la automatización de los diferentes procesos, así como la planificación, gestión y trazabilidad de los recursos [1] [2]. Un *ERP* suele tener una arquitectura modular tal y como puede apreciarse en la Figura 1, de manera que cada "modulo" cuenta con una única responsabilidad de la cual pueden beneficiarse el resto de los módulos mediante una comunicación de datos centralizada.

Figura 1. Ejemplo simplificado de una posible distribución genérica de módulos de un ERP [3].

Existe otro tipo de módulos que son transversales al resto, como suelen ser seguridad, listados e informes, o en algunos casos telemetría, por nombrar algunos. Este tipo transversal de módulos se denominan de tal forma porque son compartidos y usados por el resto de los módulos principales (por ejemplo, pero no limitados a, los ya mostrados en la Figura 1), de manera que están pensados para ofrecer una base funcional común y trabajar en base a información compartida entre diferentes módulos.

En nuestro caso, los listados e informes, que en inglés se denominan únicamente *Reports*, permiten a los usuarios de un *ERP* obtener información relevante contenida en dicho sistema para mostrarla de forma unificada y visual, de manera que apoye a la toma de decisiones y que dicha información sea también de utilidad fuera de los límites del sistema. Para este fin, existen diferentes herramientas de "*reporting"* como pueden ser *DBxtra*[1], *Acumatica Reporting*[2], *Crystal Reports*[3]o *Microsoft SQL Server Reporting Services*[4], pero la mayor problemática que presentan

[1] https://dbxtra.com/

[2] https://www.acumatica.com/cloud-erp-software/reporting-dashboards-and-data-analysis/acumatica-reporting/

[3] https://www.crystalreports.com/

[4] https://docs.microsoft.com/en-us/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-ver15

es que requieren de una buena base de conocimiento técnico para poder diseñar *Reports*, no suelen ser especialmente *user-friendly* [4], y además su coste por licencia puede llegar a ser bastante alto.

Este trabajo ha sido desarrollado en el contexto de prácticas en una empresa de desarrollo de software para el sector sociosanitario, más concretamente en el trabajo asociado al desarrollo de un microservicio de *Reports* en el departamento de I+D+i, donde se está desarrollando un *ERP* de nueva generación. En esta empresa surgió hace tiempo la iniciativa de desarrollar una herramienta de *reporting* interna que permitiese a los usuarios que no cuenten con extenso conocimiento técnico diseñar *Reports* de forma rápida, sencilla y para cualquier entidad del dominio. Estos usuarios no serían simplemente los usuarios de la aplicación, sino también los propios analistas del equipo de desarrollo y personal de soporte, permitiendo de esta forma modelar, por ejemplo, escalas médicas en forma de Informes u otros *Reports* a petición por parte de los clientes. Este objetivo fue el que se tenía en mente cuando se inició el desarrollo de la herramienta de *reporting* de nueva generación de la empresa.

Tiempo más tarde, al inicio del periodo de las prácticas de empresa durante las que se desarrolló el presente trabajo, las funcionalidades existentes de dicha herramienta de *reporting* y en especial del diseñador de Plantillas todavía no terminaban de satisfacer las necesidades de los clientes ni del propio equipo de desarrollo. Por ello, se decidió abordar diferentes mejoras del *backlog* de esta parte del *ERP*. Algunas de estas mejoras consistían en:

- Dar soporte a Secciones Repetibles para las colecciones de *Data Transfer Objects* (*DTOs*) de las diferentes entidades del dominio.
    - Diferentes formatos: *card* o tabla.
- Permitir que los Campos del *DTO* de datos puedan ser opcionales para soportar seguridad en *reporting* a nivel de Campos.
- Dar soporte a localización de Plantillas y generación de *Reports*.
- Ocultar las operaciones intermedias con los comandos de deshacer y rehacer para inserciones de Campos de la Plantilla.
- Mejoras menores de estilo y visuales: *Grid* del *TreeView* de datos redimensionable, mostrar iconos para campos de cierto tipo del *DTO* de datos de entrada, etc.

1.2 Conceptos relevantes en reporting, Listados e Informes

Para continuar, en este apartado se expondrán los conceptos más relevantes en *reporting* y para el presente trabajo.

Entidades y Modelo de Dominio

Siguiendo la terminología de diseño dirigido por el dominio [5], una entidad representa un objeto con una identidad definida e inequívoca, que trasciende el tiempo y distintas representaciones. De esta forma, una entidad puede tener diferentes representaciones o proyecciones, que dependerán de qué parte o perspectiva de la misma se quiera representar.

En el contexto de *'Clean Architecture'* [6], las entidades se definen como encapsulaciones de reglas esenciales del negocio (*"Critical Business Rules"*), que pueden ser usadas en diferentes aplicaciones en la organización, y que constituyen la base de esta, dado que son las partes con menor tendencia a cambiar cuando lo hacen las reglas del negocio.

De las dos definiciones anteriores, podemos resumir en que una entidad representa un concepto u objeto relevante para el dominio, que tiene varias representaciones en forma de proyecciones dependiendo del interés de la parte del negocio que las usa, y que contienen las reglas o lógica de más alto nivel. Por concretar, y en el contexto de un *ERP* del ámbito sociosanitario, algunos ejemplos de entidades podrían ser los Residentes, las *Plantilla*s de *Reports*, los propios *Reports* o el concepto de "presión arterial".

Las diferentes entidades de un dominio suelen estar relacionadas entre sí, conformando lo que se conoce como modelo de dominio. Así, un modelo de dominio sería *"un modelo conceptual del dominio que incorpora tanto comportamiento como datos"* [7]. Por ofrecer otra definición, en términos de *Domain Driven Design* [5], un modelo de dominio se define como *"una organización rigurosa y selectiva de abstracción del conocimiento"*.

Continuando con ejemplos del ámbito sociosanitario, podríamos decir que un residente cuenta con una serie de mediciones de su presión arterial para diferentes fechas. En otras palabras, una entidad, el residente, tiene un conjunto de presiones arteriales con fechas concretas, es decir, se relaciona con otra entidad (con diferentes instancias de esta) que tiene una serie de propiedades asociadas (la fecha en la que fue tomada). Gráficamente, esto puede observarse en la Figura 2.

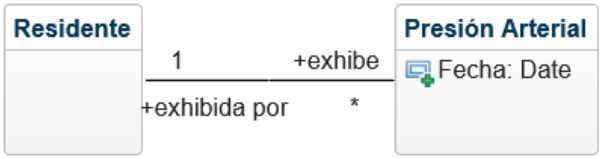

Figura 2. Relación entre la entidad Residente y la entidad Presión Arterial.

Así, el modelo de dominio de *Reports Designer* sería el que se puede observar en la Figura 3.

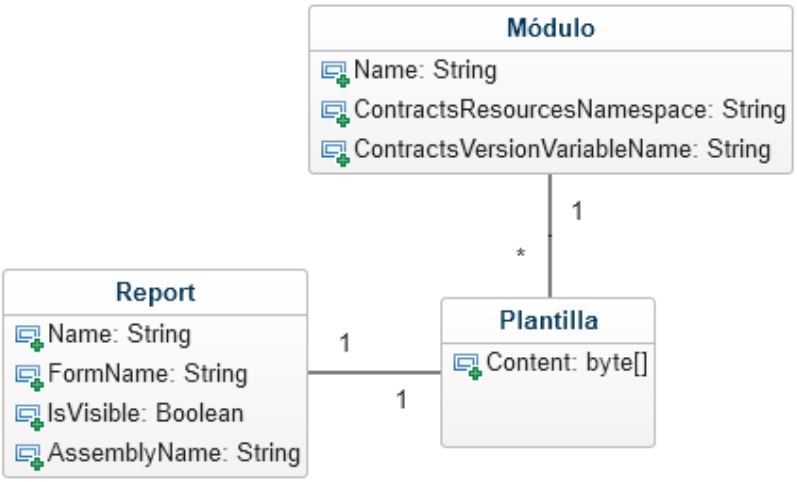

Figura 3. Diagrama del modelo de dominio de Reports Designer.

Plantillas

En el contexto de *reporting*, una Plantilla es un documento que contiene una serie de variables o *placeholders* que permiten hacer un diseño prototípico de cómo se estructura y define un Listado o Informe. Por ello, una Plantilla contiene partes fijas que no cambian, así como otras variables que, a la hora de generar un listado o informe específico (para unos datos de entrada y configuración concreta), toman el valor de los datos según les corresponda. Una Plantilla siempre está asociada a un listado o informe.

Listados e informes

El término en inglés '*Reports*' aúna los conceptos tanto de Informes como Listados. Un Informe constituye un conjunto de datos referenciados mediante Campos de una entidad o conjunto de estas diseñados de una forma concreta en una Plantilla. Un Listado conforma una especie de informe, con la diferencia de que los Campos referenciados son sobre un conjunto de entidades, repitiéndose la sección diseñada para cada elemento de dicha colección. Por ello, a lo

largo de este trabajo será común que cuando se haga referencia de forma indistinta a ambos conceptos, se use *Reports* para generalizar.

1.3 Objetivos

El principal objetivo del presente trabajo es la de implementar una serie de mejoras relevantes sobre el Diseñador de Plantillas que se encuentra en desarrollo en el ámbito del *ERP* antes mencionado.

Las mejoras que se realizarán en el diseñador de Plantillas son:

- Ofrecer la inserción y procesado de Secciones Repetibles sobre colecciones de *DTO* para las diferentes entidades del dominio que puedan ser diseñadas en diferentes formatos: como *card*, simplemente una región que se repite para las distintas ocurrencias de la colección de datos de entrada; o como tabla, con una cabecera única para la Sección Repetible que nombra las distintas columnas de las diferentes ocurrencias.
- Dar soporte a la localización y traducción de Plantillas, para que los diferentes identificadores de Elementos con los que se modelan en las Plantillas puedan ser reemplazados por sus respectivas propiedades según corresponda y localizados a la Cultura especificada [8] (español de Argentina, español de Brasil, ingles de Estados Unidos, ingles británico, etc.).
- Ocultar las operaciones intermedias con los comandos de deshacer y rehacer para inserciones de Campos de la Plantilla: en la inserción en la Plantilla de Campos del *DTO* de datos de entrada, se llevan a cabo una serie de operaciones intermedias que se ocultan al usuario del Diseñador; sin embargo, al deshacer o rehacer cambios en el documento, esas operaciones intermedias se acababan mostrando al usuario.
- Mejoras visuales como hacer el *grid* del *TreeView* de datos redimensionable, mostrar iconos para Campos de cierto tipo del *DTO* de datos de entrada o hacer que el estilo del cuadro de diálogo de los mensajes de confirmación y error en el guardado de Plantillas sean homogéneos con el estilo del control de edición de documentos.

1.4 Estructura del trabajo

En el capítulo “2. Estado del arte”, se expondrán las herramientas internas de la empresa, así como las del mercado en general, y el estado actual del nuevo Diseñador de Plantillas unificado que surgió para suplir las necesidades sobre el módulo. Finalmente, se darán las conclusiones al respecto, junto con la motivación que llevó a la empresa a continuar añadiendo nuevas características y funcionalidades al nuevo Diseñador de Plantillas unificado.

En el capítulo “3. Tecnologías empleadas”, se mencionarán las principales tecnologías empleadas en el desarrollo del proyecto, así como algunas menciones relevantes sobre el contexto tecnológico en el que se encuentra el propio Diseñador de Plantillas.

En el capítulo “4. Propuesta de módulo Diseñador de *Plantilla*s” se detallará nuestra propuesta del módulo del Diseñador de Plantillas, donde también se comentará la metodología de trabajo, exponiendo el plan de trabajo y las entregas acordadas. De igual forma, se explicará en detalle la especificación de requisitos, diseño y arquitectura, desafíos de programación y pruebas realizadas sobre este módulo. Por último, se mostrarán y explicarán los resultados obtenidos de dicha propuesta.

El trabajo finalizará con las conclusiones obtenidas del desarrollo de las mejoras para el Diseñador de Plantillas comentando cuáles podrían ser los siguientes pasos en la mejora del producto o trabajos a futuro sobre el mismo.

2. Estado del arte

Para poder exponer y reflexionar con mayor exactitud sobre el alcance y contenido del proyecto, conviene mencionar cual es el estado actual en el que se encuentran las herramientas de *reporting* en la empresa. A continuación, se pasará a ampliar el foco para comentar las herramientas de *reporting* más relevantes del mercado, para finalmente, dar unas breves conclusiones.

2.1 Estado actual de las herramientas de *reporting* de la empresa

En la versión actual del *ERP* de la empresa, en el apartado de *reporting* se usa *Crystal Reports* como diseñador de Plantillas interno al equipo de desarrollo, y al usuario se le ofrece un diseñador basado en *Microsoft Word*. Este último, es también utilizado por el equipo de soporte tanto para resolver dudas al respecto como para el diseño de algunas Plantillas de escalas y documentos requeridos por los clientes.

2.1.1 Diseñador de *Plantillas* interno: Crystal Reports

Empezando por el diseñador interno de *Crystal Reports*, en la Figura 4 puede observarse la *preview* del diseño de una Plantilla; en la parte derecha, puede apreciarse la pestaña 'Field Explorer', con los campos de la fuente de datos seleccionada disponibles para ser insertados.

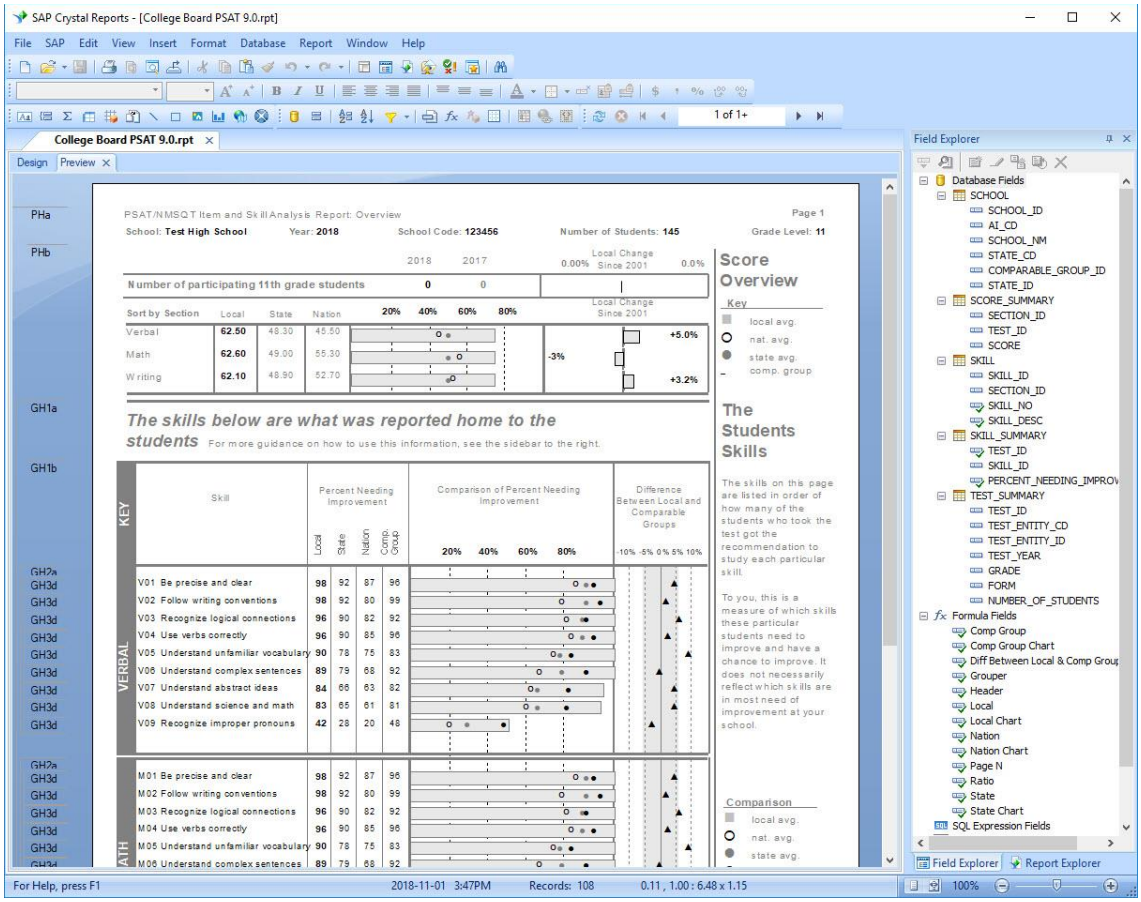

Figura 4. Pestaña 'Preview' de un report en Crystal Reports.

Esta aplicación de *business intelligence*, especializada en el diseño de Plantillas y generación de *Reports* y actualmente en propiedad de *SAP* [9] [10], ofrece gran potencia en términos de diseño y maquetado, además de exportar en diferentes formatos [11] y nutrirse de diferentes fuentes [12] en su versión de 2020. Sin embargo, la complejidad que envuelve el diseño de una Plantilla (dado que se trata de una herramienta orientada a personal con un conocimiento técnico profundo), hace que el esfuerzo necesario para trabajar con ella sobrepase los beneficios que

otorga al *ERP* de la empresa. En concreto, el equipo del departamento de desarrollo encargado del uso de este diseñador de Plantillas a nivel interno encuentra algunos problemas al respecto de su uso[5]:

- Bajo rendimiento en uso de sub-informes y cuando se trabaja con múltiples fuentes de datos, especialmente involucrando en ambos casos el formato *TTX* [13] [14]. Además, también se mencionan algunos problemas con la liberación de recursos de memorial.
- Problemas de cambios al actualizar la versión: cambios no retro compatibles que generaban inconsistencias en fórmulas utilizadas en Plantillas diseñadas previamente.
- Problemas puntuales en la impresión para distintas impresoras, mostrando inconsistencias en la visualización de listados. Principalmente, ocurre porque *Crystal Reports* hace uso de los *drivers* de la impresora destino.
- Apariciones puntuales de mensajes pocos descriptivos: *"Este campo es desconocido"*, sin indicar qué campo ni dónde se encuentra o diálogos con mensajes vacíos y únicamente el botón de "OK".
- Perdidas puntuales de modificaciones en fórmulas en el guardado, sobre todo en el caso de condiciones de supresión de campos.
- Inconsistencias en el proceso de diseño debidas a diferentes configuraciones del entorno.
- Escasa o nula personalización de la generación automática de consultas *SQL*.
- Problemas en la visualización de textos con cadenas de caracteres extensas y en varias líneas.

Además, conviene mencionar que cada equipo que haga uso de este requiere de una compra de licencia con un coste de compra de 579.59 EUR[6] en España [15]. Es decir, cada equipo necesita de una licencia de compra única.

Otra de las desventajas que presentaba esta herramienta, era que no soportaba la seguridad al nivel que se requería del motor de *Reports*, no permitiendo trabajar a nivel de *DTO* o campos y con permiso sobre ellos. Además, *Crystal Reports* no es una herramienta que tenga un soporte del todo correcto para *Reports* que hacen referencia a una sola entidad (informes: contratos, documentos de Régimen Interior, informe de transportes, etc.), ya que está más pensada para listados. En términos de pruebas, también se encontraba que, al ser propietaria, solo permite pruebas de UI o "*end2end"* con los problemas que ello conlleva (fragilidad al depender fuertemente del *layout* y no de la propia funcionalidad en sí, además de tener múltiples dependencias con otros servicios; alto tiempo de ejecución; falsos positivos; etc.) [16] [17].

2.1.2 Diseñador de *Plantilla*s externo: basado en Word

Al usuario del *ERP* se le expone un diseñador de Plantillas basado en *Microsoft Word*. Es por esto último que dicho diseñador obliga al usuario a tener una licencia de *Microsoft Office* activa, lo que conlleva un coste mínimo de 10.50€ por usuario al mes para la versión *Microsoft 365 Empresa Estándar* [18] en España. En este caso, los clientes deben de contar con dicha licencia para poder hacer uso en primer lugar del diseñador de Plantillas en cuestión.

Conviene mencionar que, si bien es uno de los módulos más utilizados por los clientes, actualmente está perdiendo uso ya que se encuentra limitado respecto de informes que realicen operaciones del tipo: totales, ratios, indicadores, etc. Los casos en los que los clientes suelen

[5] Esta información se obtuvo mediante una recopilación de problemas e inconvenientes comunes entrevistando a algunos miembros del equipo de desarrollo.

[6] Versión SAP Crystal Reports 2020 64-bits para un solo usuario.

encontrar de más utilidad este módulo son para complementar formularios personalizables por el centro (contratos de ingreso, autorizaciones, fe de vida, etc.). Por el contrario, listar u obtener información estadística resulta complicado, y además en la mayoría de las Plantillas, el cliente acaba delegando en el departamento de soporte el diseño de aquellas que salgan de los tipos de documentos anteriormente mencionados. Algunos problemas relacionados con Plantillas de cierta complejidad[7]:

- La aplicación de filtros por nodos es propensa a fallos.
- El diseño entre culturas[8] (p.ej. de catalán a castellano) hace que funcione en una pero no en otra.
- Los datos de tablas anidadas no se recogen correctamente.

Además, otros problemas acerca el diseñador de *Reports* basado en *Word*[9] que fueron reportados por el equipo de soporte del *ERP:*

- Se estima que el 80% de los datos registrados por las organizaciones de los clientes pasan por el módulo de *Reports*. Además, todos los días se reciben varias llamadas al respecto del diseñador o generación de *Reports.*
- Limitado en términos de tablas de entidades: límite de máximo de dos tablas para una Plantilla y carece de aviso o información al respecto. Esto genera un mensaje de fallo poco descriptivo al final del proceso de selección de datos de entrada de la Plantilla, haciendo que se pierda la configuración que el usuario había elegido.
- Para el cliente medio, el diseñador es de gran complejidad y suelen acabar pidiendo que las Plantillas se diseñen por la propia empresa. Tanto al propio equipo de soporte como a los clientes, les gustaría que el diseñador fuese mucho más simple.
- Dada la complejidad ya mencionada, se requiere de una formación para que el cliente pueda ser capaz de diseñar una Plantilla de forma básica.
- Los clientes suelen recriminar que los *Reports* impresos ocupan mucho espacio, tanto en la propia hoja como en el número de estas. Esto se debe a que la información mostrada es total y no se pueden generar proyecciones configurables. Lo ideal sería que la información pudiera ser discriminada en base a los permisos de quien imprime el *report* y la configuración escogida en tiempo de impresión; de esta forma, solo se mostraría información relevante para quien vaya dirigido el *Report* en cuestión.
- Con relación al punto anterior, el hecho de mostrar toda la información modelada en una Plantilla a la hora de la impresión representa un gran problema de seguridad: si un rol en la organización no tiene permisos para consultar cierta información, si esta se encuentra modelada en un *Report*, entonces queda expuesta y puede ser consultada en el *Report* impreso.
- Un trabajo común para el equipo de soporte es el mantenimiento de Plantillas que quedan desactualizadas, tanto por cuestiones de negocio como a nivel tecnológico del funcionamiento interno del diseñador.
- Existen problemas relativos a la fuerte dependencia de detalles de implementación sobre el comportamiento de la generación de *Reports:* la ordenación de los datos afecta a como se generan las consultas *SQL*. Por ello, a la hora del diseño de

[7] Esta información sobre el uso y problemáticas del diseñador de *Plantilla*s basado en *Word* se obtuvo de conversar directamente con la encargada del departamento de comercial sobre la percepción de los clientes y uso que daban del propio diseñador.

[8] Entendemos el término 'Cultura' tal y según se explica en [8].

[9] Esta información se obtuvo de una reunión con un miembro del equipo de soporte, donde se comentaron los diferentes puntos mencionados.

Plantillas con cierta complejidad, es imperativo conocer estos detalles y manejar fallos asociados a los mismos.

- Algunos *Reports* generados requieren de un tratamiento manual en un procesador de hojas de cálculo que permita pivotar y formatear datos, ya que en la mayoría de los casos estos quedan algo ilegibles.

En resumen, para el módulo de *reporting* del *ERP* en producción se requieren de dos herramientas distintas: pagando la licencia de una por parte de la empresa y de la otra por parte de los clientes, teniendo que conciliar el diseñador basado en *Word* con los cambios no-regresivos que se hagan sobre la herramienta de *Crystal Reports* y en general obligando a tener personal con conocimientos en ambos. Esto obliga a que, por ejemplo, alguien del departamento de soporte requiera tener un conocimiento técnico, pero también conocimiento del uso del diseñador de Plantillas basado en *Word* para dar soporte al cliente en caso de problemas o cuestiones respecto de este.

2.1.2.1 Ejemplos de Plantillas diseñadas en el Diseñador de Plantillas externo

En este apartado, se mostrarán algunos ejemplos de Plantillas que han sido diseñadas por el equipo de soporte bajo petición de clientes. Estas Plantillas resultan de gran interés para los clientes, ya que son parte del conjunto de Plantillas que más utilizan y sobre las que más consultas llegan al departamento de soporte. Dichas Plantillas de ejemplo pueden observarse en la Figura 5, Figura 6, Figura 7, Figura 8 y Figura 9.

DATOS PERTINENTES – DIETA DE RESIDENTE

Nombre: Nombre **Habitación:** Código Habitación

Seguimiento – Dietista

Usuario: Usuario

Fecha/Hora: Fecha/Hora

Descripción: Descripción

Datos – Dietista

Ayuda para comer: Ayuda para Comer

Horario asignado para las comidas: Horario Comidas

Personal asignado para las comidas: Personal Asignado Comidas

Tiempo asignado para las comidas: Tiempo Asignado Comidas

Preferencias alimentarias: Preferencias Alimentarias

DIETA

Fecha Desde	Fecha Hasta	Dieta	Textura	Observaciones
Fecha Desde	Fecha Hasta	Dieta	Textura	Observaciones

ALERGIAS ALIMENTARIAS

Alergia Alimentaria	Fecha Desde	Fecha Hasta	Observaciones
Alergia Alimentaria	Fecha Desde	Fecha Hasta	Observaciones

ALIMENTOS RESTRINGIDOS

Alimento Restringido	Fecha desde	Fecha hasta	Observaciones
Alimento Restringido	Fecha desde	Fecha hasta	Observaciones

Figura 5. Ejemplo de Plantilla "Datos pertinentes – Dieta de Residente".

MEDICACIÓN ACTUAL – SEGUIMIENTO MÉDICO

Residente: Nombre **Habitación:** Código Habitación

Último seguimiento médico

Médico: Médico **Fecha:** Fecha/Hora

Motivo consulta: Motivo Consulta

Diagnóstico: Diagnóstico

TRATAMIENTO ACTUAL

Médico: Médico **Fecha:** Fecha

Medicamento	Patología	MAD	DES	COM	MER	CEN	NOC	Pau. Esp.	Fecha Final	Fecha Inicial	Vía Adm.	Obs.
Medicamento	Patología	Dosis 04h	Dosis 08h	Dosis 12h	Dosis 16h	Dosis 20h	Dosis 24h	Pauta Especial	Fecha Final	Fecha Inicial	Vía Administración	Observaciones

Figura 6. Ejemplo de Plantilla "Medicación actual – Seguimiento Médico".

PRUEBAS DIAGNÓSTICAS – RESIDENTES

Nombre	Fecha Nacimiento	Nº de Identificación	Descripción Tipo Prueba	Fecha Realización	Fecha Recepción	Presencia Anticuerpos Genéricos	Presencia Anticuerpos IgG	Presencia Anticuerpos IgM	Presencia Virus COVID-19	Observaciones
Nombre	Fecha Nacimiento	Nº de Identificación	Descripción Tipo Prueba	Fecha Realización	Fecha Recepción	Presencia Anticuerpos Genéricos	Presencia Anticuerpos IgG	Presencia Anticuerpos IgM	Presencia Virus COVID-19	Observaciones

Figura 7. Ejemplo de Plantilla "Pruebas diagnósticas – Residentes".

RESIDENTES PERNOCTA ENTRE 65 AÑOS Y 70 AÑOS

Nombre	Fecha de Nacimiento	Edad	Estancia
Nombre	Fecha Nacimiento	Edad	Estancia

Figura 8. Ejemplo de Plantilla "Residentes pernocta entre 65 años y 70 años".

Nombre: Nombre **Habitación:** Código Habitación

Tratamiento Actual

Fecha: Fecha

Medicamento	Fecha Final	Fecha Inicial	04h	08h	12h	16h	20h	24h	Observaciones
Medicamento	Fecha Final	Fecha Inicial	Dosis 04h	Dosis 08h	Dosis 12h	Dosis 16h	Dosis 20h	Dosis 24h	Observaciones

Figura 9. Ejemplo de Plantilla "Tratamiento actual".

2.1.3 Diseñador de Plantillas unificado

Dado que el estado actual de las herramientas internas no acababa de satisfacer las necesidades de la empresa y los clientes con respecto del diseño de Plantillas, se inició el desarrollo de un Diseñador de Plantillas de nueva generación que unificase el Diseñador interno y externo. Por ello, en el presente apartado expondremos el diseño y arquitectura de dicho Diseñador de Plantillas unificado, exponiendo tanto aspectos externos relativos al contexto en el que este se encuentra como aspectos internos del propio Diseñador.

2.1.3.1 Arquitectura del microservicio de Reports Designer

En este apartado se explicará el caso concreto del microservicio de *Reports Designer*, el *backend* del Diseñador de Plantillas.

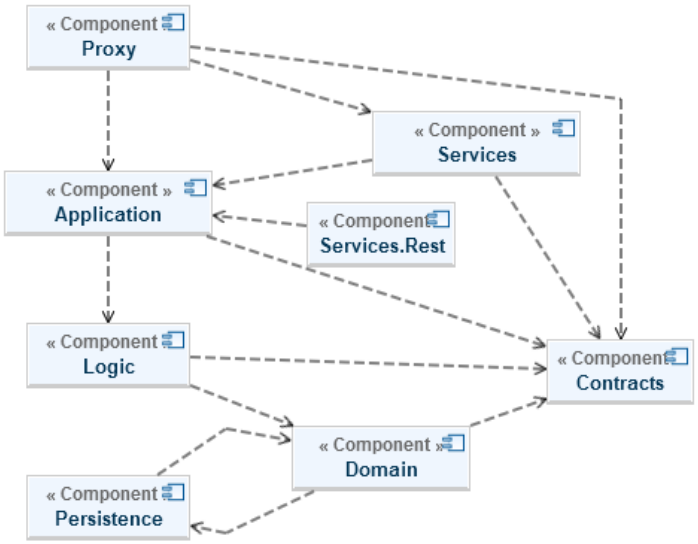

Figura 10. Vista general de las capas del microservicio de Reports Designer.

En la Figura 10 se muestra un resumen de la arquitectura de este microservicio, que se irá explicando brevemente a continuación.

La capa de contratos contiene las interfaces que especifican las acciones que el microservicio puede realizar, así como *DTOs*, recursos o demás artefactos de interés para los posibles clientes del mismo. En otras palabras, sería el contrato público que expone el microservicio.

La capa de aplicación con tiene el código autogenerado que orquesta las llamadas a la capa de lógica. Así, también se hacen algunas comprobaciones de permisos y validaciones sobre los argumentos con los que se realizan las llamadas.

La capa de dominio contiene las entidades del dominio relevantes para el contexto del propio microservicio. Para clarificar esto último, mencionar que en términos de *Domain Driven Design* [5], se habla de que en productos especialmente grandes donde el modelo de dominio es, de igual forma, efectivamente grande, aparece el término de *bounded context* (contexto delimitado). Estos contextos permiten hacer uso de proyecciones parciales sobre el modelo de dominio en su totalidad, permitiendo que, para cada contexto concreto, se usen aquellas entidades o conceptos relevantes para el mismo. Así, se permiten estos *bounded context* definiendo sus relaciones con otros contextos, con el objetivo de aunar todo el conocimiento del modelo de dominio, pero permitiendo usar proyecciones para que el foco de atención esté sobre lo importante en cada contexto. Por último, comentar que la capa de dominio contiene la lógica asociada a dichas entidades de dominio (reglas de negocio, que suelen variar poco o nada) y validaciones sobre ellas.

La capa de lógica, es la encargada de encapsular toda la lógica de negocio que implementa los diferentes casos de uso del microservicio. En ella se encuentran las implementaciones de los *endpoints* que se comentaba anteriormente, además de otras clases de utilidad como *queries* predefinidas sobre los repositorios de las entidades de dominio o el *TemplateBuilder*, encargado de resolver los enlaces a otros *Reports* (más conocidos como *Subreports*) y construir finalmente la Plantilla.

La capa de persistencia recoge toda la lógica asociada a la persistencia del microservicio. Por ello, conoce cómo persistir cada entidad de la capa de dominio.

La capa de proxy es la encargada de configurar la conectividad de *Reports Designer* con la que se invocará a la lógica del mismo. En base a la configuración (online, *in-process* o hibrido en base a la calidad de la conexión), se hará una llamada *HTTP* sobre los controladores de la capa de servicios, se redirigirá la llamada directamente a la propia capa de aplicación o una mezcla de ambas.

La capa de servicios es la encargada de implementar los controladores *REST* con las que se atenderían las peticiones de los usuarios en el modo online, y de implementar los archivos de configuración que se requieren para conectar el resto del microservicio con tecnologías externas al mismo (principalmente, se encarga de hacer la implementación mínima para que pueda ser alojado como una aplicación *ASP.NET Core*[10] o configuraciones de *Docker*[11] o *SignalR*[12]). La denominada *"Services.Rest"* sería su equivalente pero que implementaría en sus controladores única y exclusivamente operaciones *Create Read Update Delete (CRUD)* sobre las entidades del dominio.

Por último, conviene mencionar dos puntos sobre el microservicio de *Reports Designer*. Por un lado, resaltar que esta arquitectura e infraestructura es autogenerada por el modelo de aplicación específico modelado en las *DSL Tools*, y que es casi idéntico al del resto de microservicios del *ERP* en desarrollo. Por otro lado, en ningún momento se ha hablado de la generación como tal del *Report* en base a su Plantilla asociada; esta responsabilidad pertenece al motor de *Reports*, el microservicio de *Reports Engine*, que simplemente se encarga de reconocer y remplazar los *tags* encontrados en la Plantilla por los datos concretos para los que se quiere generar el *Report* en cuestión. Tanto el modelado y generación de código como todo lo que envuelve al motor de *Reports* (microservicio *Reports Engine*), queda fuera del alcance de este trabajo. Por ello, solo se mencionarán los detalles que sean realmente importantes para dichos temas en cuestión y de forma muy breve.

2.1.3.2 Arquitectura del formulario WPF del Diseñador de Plantillas

El Diseñador de Plantillas se encuentra implementado como un formulario en *WPF*[13], con una arquitectura *Model View ViewModel (MVVM)*. Este patrón arquitectónico busca la separación completa de la vista (siendo ésta en el caso del módulo del Diseñador de Plantillas un fichero *XAML*) del modelo (de dominio) mediante el uso de un mediador [19]. Este mediador sería el *view model*, que se encarga de exponer comandos que orquestan la lógica necesaria para interactuar con el modelo, y a los que la vista pueda acoplarse. Este enlazado de los *endpoints* del *view model* con los componentes y controles del formulario de la vista *(data bindings)*, se hace de forma automática mediante un *binder*[14]. De esta forma, se consigue que la vista sea únicamente responsable de organizar visualmente los elementos gráficos (componentes, controles, etc.) para los distintos formularios; las acciones del usuario sobre la vista se redirigen directamente a los comandos enlazados del *view model;* el *view model* orquesta la lógica sobre el modelo para tramitar la petición; por último, los datos o información que deba actualizar de nuevo la vista, hará el camino inverso hasta llegar a la propiedad de la misma que deba actualizar.

---

[10] Para información al respecto, consultar [84].

[11] https://www.docker.com/

[12] Para información al respecto, consultar [85].

[13] Tanto *WPF* como sus principales características, se expondrán con mayor detalle en la sección "3.2 Windows Presentation Foundation".

[14] Esta funcionalidad de *binding* implícito automatizado, viene dada por el *XAML* de la vista, permitiendo especificar declarativamente dichos enlaces. Para más información, consultar [41] y [42].

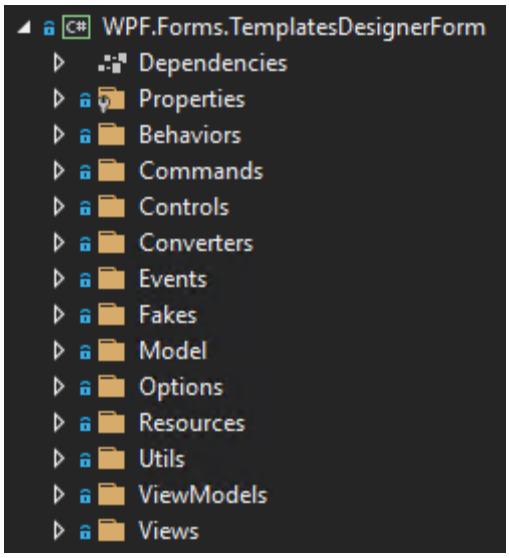

Figura 11. Vista general de los directorios del proyecto del formulario del Diseñador de Plantillas.

En la Figura 11 podemos apreciar un resumen de la arquitectura del Diseñador de Plantillas.

En este caso, en el directorio *ViewModels* existe un único *ViewModel* que trabaja con la única vista (formulario) del proyecto y que, de forma similar a como se ha comentado anteriormente, se encarga de gestionar las peticiones que llegan desde los *bindings* con la vista para llevar a cabo los casos de uso. En él, se definen los métodos a invocar para diferentes comandos, como es el caso del comando encargado de cargar la Plantilla actual, el de insertar el Elemento seleccionado en la Plantilla, persistir la Plantilla o el comando encargado de orquestar el cierre de la aplicación. Remarcar que, en el caso del comando de guardado de la Plantilla, se orquesta una cierta lógica y tratamiento sobre la Plantilla antes de delegar la invocación del comando sobre la implementación por defecto del Control del editor del documento.

El formulario *XAML* de la vista, se encuentra en el directorio *Views*. Los *bindings* en cuestión que tiene con el *view model* serían: el evento de cerrar el formulario, los elementos raíz que hagan de datos de entrada para el *Report* o cualquiera de los comandos mencionados anteriormente. Estos *bindings* tanto de datos como de comandos, son los que se enlazan sobre el *view model* para ejecutar la lógica asociada.

El directorio de *Behaviors* contiene los diferentes *behaviors* definidos para el formulario. Algunos ejemplos serían los que definen el comportamiento de los controles sobre los que se puede hacer click y arrastrar, o los que definen el comportamiento para poder hacer foco sobre dichos controles.

En *Commands* podemos encontrar diferentes comandos que se encargan de remplazar o customizar los comandos predefinidos del control de edición de documentos de *DevExpress*, como el comando encargado de guardar la Plantilla u otros más específicos del propio control de *DevExpress*, como por ejemplo serían los de cargar el documento o crear un nuevo documento vacío.

Por último, encontramos en el directorio de *Controls* la interfaz que define los contratos públicos del editor de documentos y correspondiente implementación, que se encarga de implementar el Control dedicado a la edición de documentos de texto rico. Para el caso de dicha implementación, se implementa el Control propietario de *DevExpress*. En esta interfaz y correspondientes implementaciones se encargan de orquestar la lógica encargada de tratar con

dicho control, es decir, la lógica que interactúa directamente con el control de terceros de edición de documentos *WordprocessingML*[15].

Para hacer uso de este formulario, existen diferentes contextos de carga que definen con qué parámetros y configuración se ejecutará. En el caso de las pruebas manuales, se hace uso del contexto de los *ManualTests*, que lanza el formulario del Diseñador de Plantillas como una aplicación independiente; para el uso del formulario integrado en las *DSL Tool* (con el que los diseñadores encargados de modelar puedan diseñar cómodamente diferentes Plantillas), se define otro contexto de carga; para su uso integrado en la interfaz del *ERP*, se plantea un contexto de carga similar al usado para las *DSL Tool*.

2.2 Estado actual de las herramientas de *reporting* del mercado

Pasaremos ahora a comentar cuales son las principales herramientas y soluciones para *reporting* del mercado. Es importante mencionar también el hecho de que, al ser considerado el *reporting* un tipo específico de *business intelligence (BI)*, muchas herramientas del mercado que se categorizan como *reporting* suelen ir integradas o ligadas a controles y funcionalidades más propias de *business intelligence* en general.

Acumatica Reporting

Acumatica Reporting es la herramienta de *reporting* del *Cloud ERP* modular de *Acumatica*[16]. Esto significa que dicho módulo de *reporting* puede ser utilizado de forma individual, con la problemática de que la integración que habría que llevar a cabo para hacerlo funcionar en otro *ERP* que no sea el propio de *Acumatica*, pueda llegar a ser algo más costosa de lo que sería con un módulo *in-process* instalado en el cliente. En la Figura 12 se observan algunos ejemplos de *Reports* generados mediante *Acumatica Reporting*.

[15] Este formato se explicará en mayor detalle en el capítulo "3.3 OpenXML SDK y WordprocessingML".

[16] https://www.acumatica.com

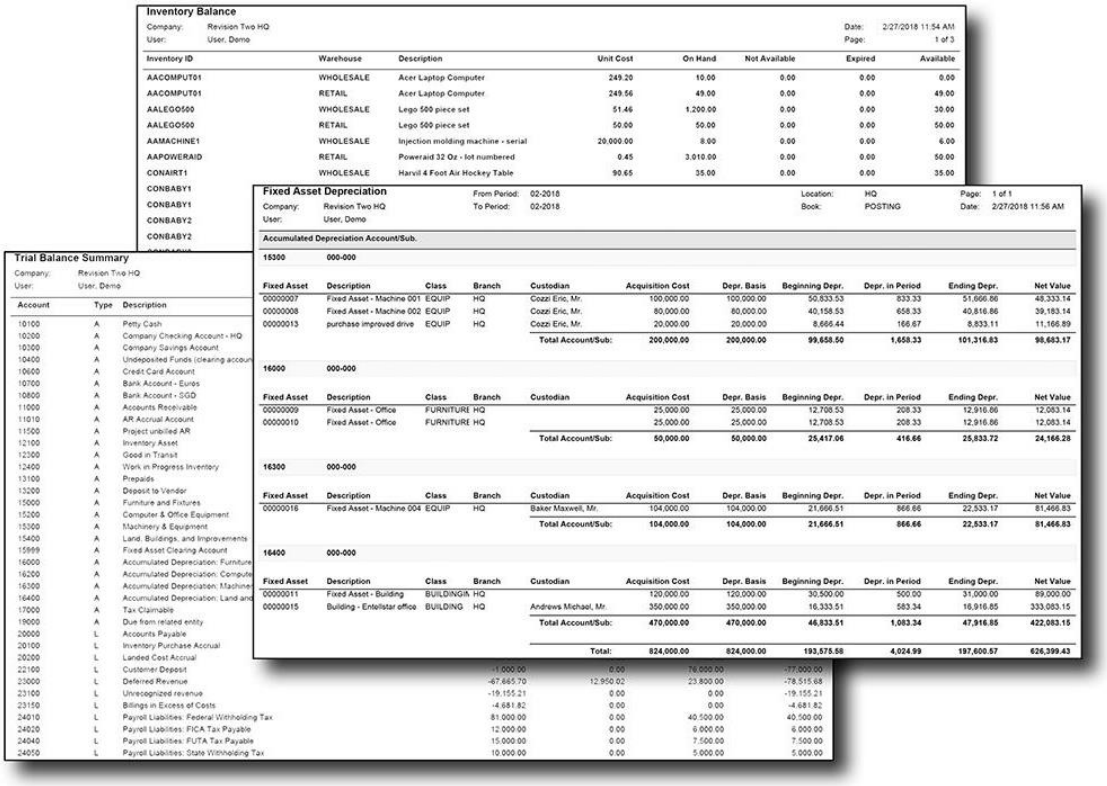

Figura 12. Ejemplos de Reports generados mediante Acumatica Reporting.

Algunas de las características de mayor relevancia que se destacan en la página principal del vendedor, son:

- Diseñador de Plantillas con soporte a previsualización.
- Enlazado de múltiples *Reports* en uno único.
- Seguridad a nivel de roles.
- Múltiples formatos de *Reports: HTML*, *PDF*, Excel, Word, etc.
- Filtros reusables para formularios.
- Las ventajas propias de un *SaaS Cloud* [20], así como sus correspondientes desventajas.
- Reports localizados dependiendo del soporte multilenguaje del usuario [21].

Phocas Software

La empresa *Phocas Software* ofrece su propia solución para integrar su herramienta de *reporting*[17] con varios *ERP* de importancia del mercado. Algunos de estos ejemplos son: *Epicor*, *SAP*, *Sage*, *Oracle* entre otros [22].

[17] https://www.phocassoftware.com/software/core/reporting

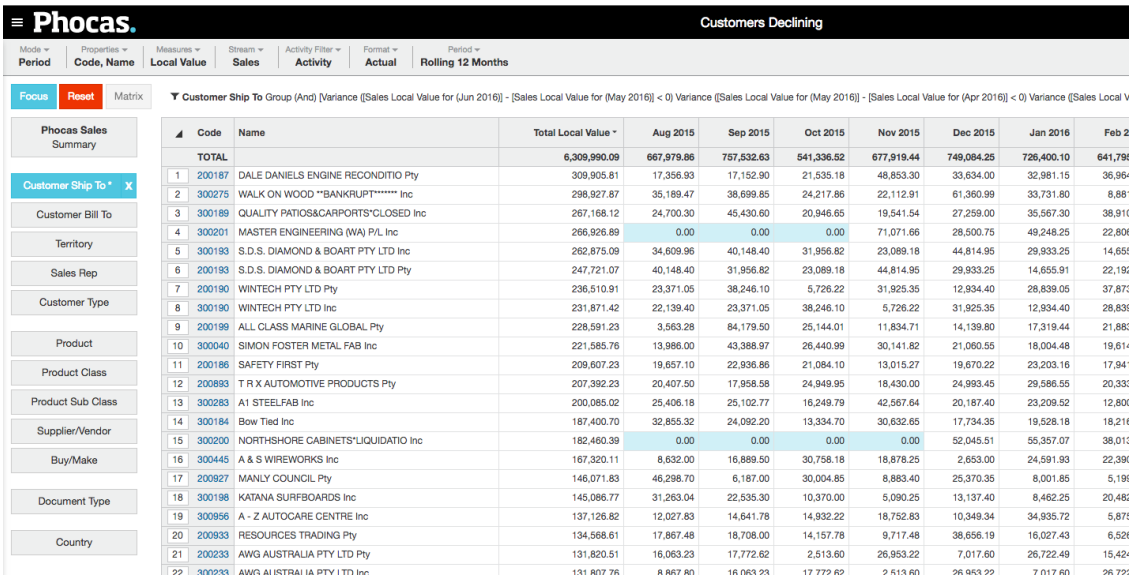

Figura 13. Ejemplo de Report en Phocas Software reporting.

Algunos beneficios que aporta esta herramienta serían:

- Generación automatizada de *Reports* y con alertas de finalización, prometiendo un tiempo bajo de procesamiento.
- Ofrece tanto la opción de uso en la nube como *on-premise*[18] (nube privada).
- Soporte a uso directo de métricas e indicadores de negocio en *Reports.*
- Ofrece integración con varios de los *ERPs* de mayor relevancia del mercado, como ya se ha comentado.

Izenda Reports

Izenda Reports[19] es otra solución que ofrece la posibilidad de mostrar la información mediante *Reports* y controles de *BI* en un *dashboard* [23]. De igual forma se trata de una solución *SaaS* accesible desde navegador web, pero también permite ser instalado *on-premise.*

[18] Instalado de forma privada y local para la organización que lo utiliza.
[19] https://www.izenda.com/dashboards-reports/#reports

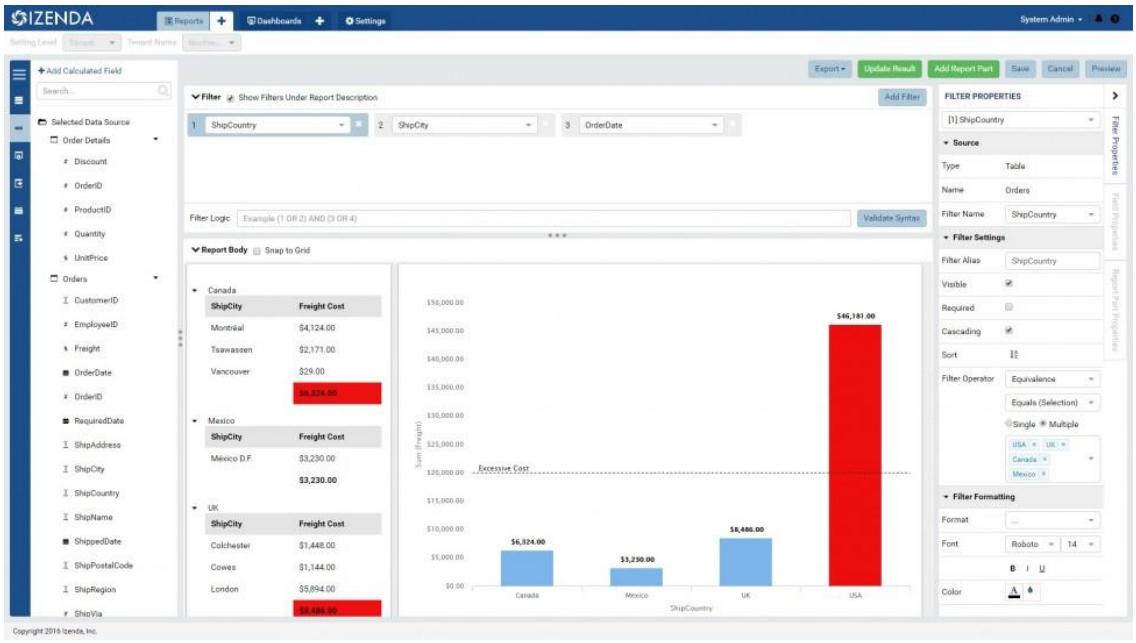

Figura 14. Ejemplo de Report en Izenda Reports.

Algunas de las características de mayor relevancia de la herramienta, serían:

- Filtros sobre *Reports.*
- Uso de gráficos y *grid.*
- Personalización y compartido de r*eports* y *dashboard* en tiempo real.

DBxtra

Respecto de las alternativas anteriores, *DBxtra*[20] aporta características muy similares sobre *reporting* ad-hoc, estando también disponible en navegador web. Al igual que ocurre con *Crystal Reports*, el acceso a datos se hace mediante consultas *SQL* y con el paradigma relacional en general.

[20] https://dbxtra.com/

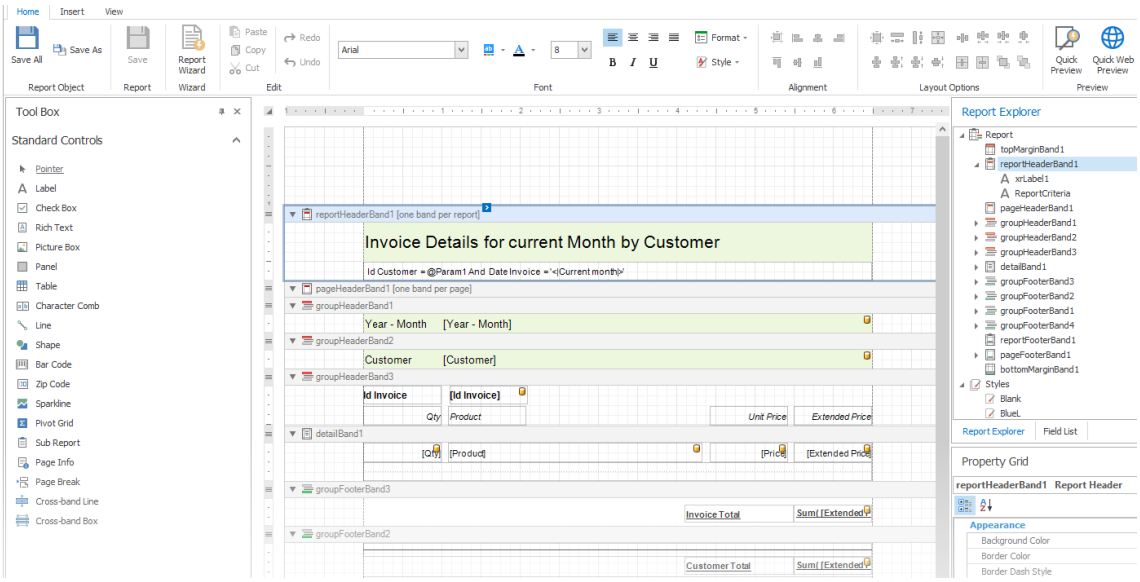

Figura 15. Ejemplo de Report en DBxtra.

Al respecto de las características más relevantes:

- Programación temporal de generación de *Reports.*
- Visualización en tiempo real de los datos en *Microsoft Excel* [24].
- Múltiples formatos de *Reports: XML*, *PDF*, *Excel*, *Word*, *CSV*, imágenes y *HTML*.
- *Reports* interactivos visualizables tanto desde servicio web [25] como de escritorio.
- Conexión con múltiples fuentes de bases de datos: *MS Access*, *MySQL*, *Oracle*, *Excel*, *ODBC*, *OLEDB*, etc.

BIRT

Una herramienta de *reporting* de código libre es *BIRT*[21]. De igual forma, ofrece integración con diferentes indicadores de *bussiness intelligence* para la visualización de datos.

[21] https://www.eclipse.org/birt/about/

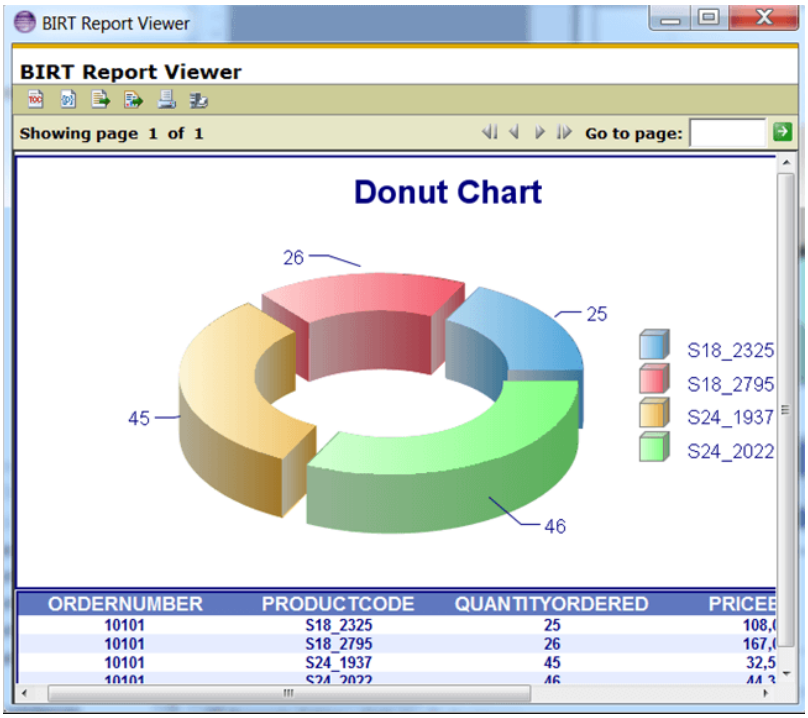

Figura 16. Ejemplo de Report en el Report Viewer de BIRT.

Algunas características de la herramienta en cuestión serían:

- Diseñador de *Plantillas "What You See Is What You Get"* (*WYSIWYG*).
- Explorador de datos que permite organizar las diferentes fuentes de datos (conexiones) y conjuntos de datos *(queries)*.
- Paleta con diferentes elementos predefinidos para el diseño de *Reports*.
- Agnóstico al sistema operativo.

Microsoft SQL Server Reporting Services

Microsoft SQL Server Reporting Services (SSRS) es una de las soluciones ofrecidas por *Microsoft* para el *reporting* basado en *SQL* [26]. De más reciente aparición, podríamos hablar de *Power BI Report Server*, que básicamente se trataría de un *superset* de *SSRS* [27]: permite hacer lo mismo, pero añadiendo una capa de funcionalidad adicional más centrada en *BI* en general que en *reporting*.

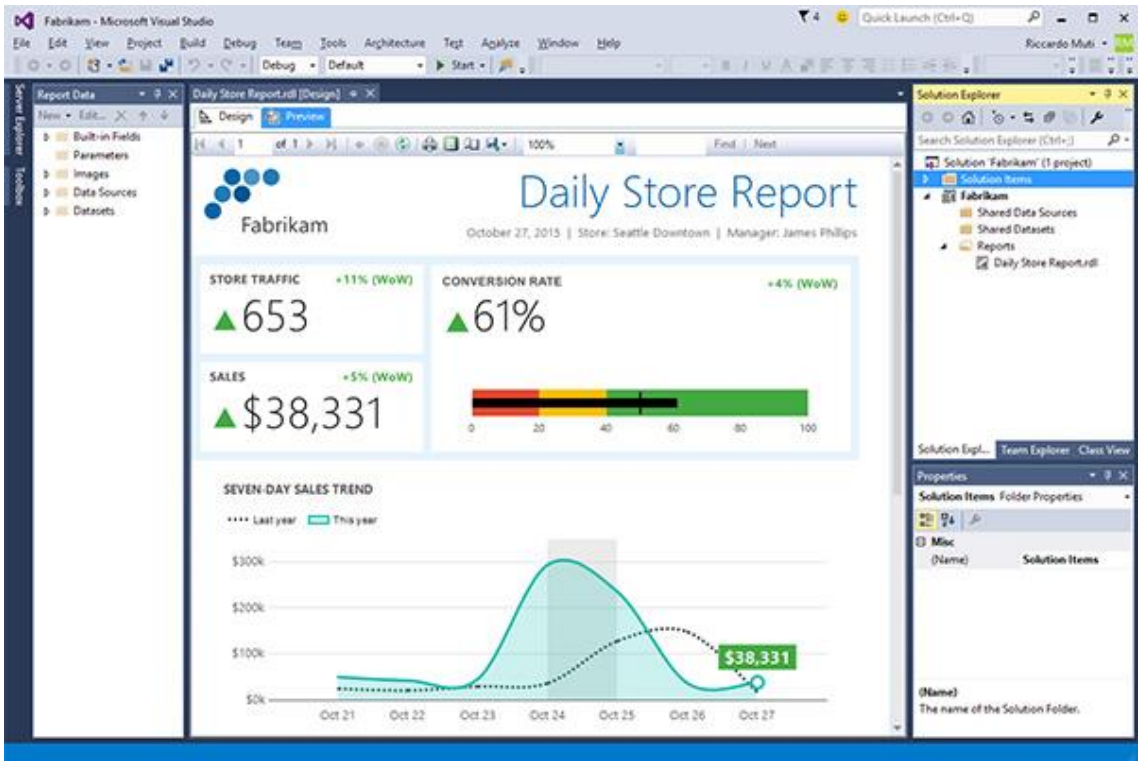

Figura 17. Ejemplo de Report en SQL Server Reporting Services.

Algunas de las características que definen la herramienta son:

- Uso de las *SQL Server Data Tools* [28] para el diseño de *Reports* "tradicionales" en formato *PDF* o *Word.*
- Uso de *SQL Server Mobile Report* [29] para el diseño de *Reports* interactivos para dispositivos móviles.
- Uso de portal web [30] para visualizar, configurar y programar *Reports*, configurar seguridad basada en roles, crear enlaces entre diferentes *Reports*, creación y gestión de *Key Performance Indicators (KPI)*, entre otras cosas.

Otras herramientas que tienen funcionalidades y características similares serían: *Solarwinds*[22], *Answer Rocket*[23] o *Board*[24].

En general, todas las herramientas comentadas ofrecen características y problemáticas similares. Tal y como se exponía en la introducción del trabajo, este tipo de herramientas suelen requerir de un cierto conocimiento técnico, y excepción de las soluciones de código libre, no permiten realizar modificaciones en su funcionalidad para adaptarlas a las necesidades de la organización. Además, aportan un coste adicional tanto al cliente final del *ERP* como a la organización que lo desarrolla.

[22] https://bit.ly/3Ae5w8o
[23] https://www.answerrocket.com/a-reporting-tool/
[24] https://www.board.com/en/reporting

2.3 Conclusión

Con lo comentado en los dos apartados anteriores, el equipo de desarrollo identificó que existía una necesidad de mejorar las prestaciones ofrecidas por el actual Diseñador de Plantillas en términos de usabilidad, facilidad de aprendizaje y uso, y de funcionalidades específicas en lo relativo a capacidades de diseñar *Reports* específicos. De igual forma, se planteó unificar el uso que se hacía de los dos Diseñadores de *Plantilla*s que convivían: el interno basado en *Crystal Reports* y el externo basado en *Microsoft Word*; aprovechando mejor el conocimiento y esfuerzo de desarrollo y mantenimiento sobre una única herramienta de *reporting.* Es por ello por lo que se decantó por la opción de desarrollar una herramienta interna que satisficiese dichas necesidades. En el punto en el que se empezó el presente trabajo, el estado de desarrollo de dicho Diseñador de Plantillas unificado todavía no termina de satisfacer las necesidades y funcionalidades que se esperaban de él; es por ello por lo que se acabaron planteando una serie de mejoras sobre el mismo que lo acercaran más al estado deseado.

3. Tecnologías empleadas

En el presente apartado expondremos los fundamentos del entorno de desarrollo y programación que soportan el Diseñador de Plantillas de *Reports* que nos atañe.

3.1 .NET Core y .NET Framework

Algunos de los entornos de ejecución disponibles para *C#* son *.NET Core* y *.NET Framework*, en conjunción a la nueva versión cinco de *.NET Core (.NET Core 5.0)* [31] [32] que busca unificar el desarrollo en *C#*[25]. Gracias a esto último, se elimina la necesidad de *.NET Standard* como nexo común entre los dos primeros a la hora de desarrollar funcionalidad común a ambos.

Hasta la salida de *.NET Core 5.0*, *.NET Standard* fundamentaba todos los frameworks de la familia de *.NET*, ofreciendo uniformidad en el ecosistema abstrayendo toda la implementación común y agnósticas a la plataforma, permitiendo desarrollar librerías que sean referenciadas tanto por proyectos *.NET Core* como proyectos en *.NET Framework.* Este modo de compatibilidad tiene ciertas restricciones: si una librería de *.NET Framework* hace uso de alguna funcionalidad o librería específica que no esté disponible para *.NET Standard*, esta compatibilidad no será posible (p.ej. *WPF*).

.NET Framework ha sido la base del desarrollo para plataformas *Windows* desde su lanzamiento a finales del 2000 [33], y dada su antigüedad y madurez, tiene una gran cantidad de librerías que hacen uso de él. Por ello y en adición, hay algunas recomendaciones sobre seguir haciendo uso de *.NET Framework* frente a *.NET Core* (tanto su versión 5 como anteriores), algunas listadas en el apartado *"When should I still use .NET Framework 4.X?"* de la referencia [34][26]

Así, *.NET Core* sería el sucesor multiplataforma de *.NET Framework* [34], ofreciendo un entorno de desarrollo, modular ligero y multiplataforma (*Windows, Linux* y *macOS*). Además, como se menciona en el propio título de [31], la intención de *Microsoft* es de que *.NET Core* sea el entorno de desarrollo por defecto para las futuras aplicaciones de .NET.

Cabe también mencionar el *framework* de *Xamarin*, centrado en el desarrollo nativo para plataformas móviles como *iOS* y *Android.* Como parte de este *framework*, se ofrece *Xamarin.Forms*, que básicamente permite desarrollar interfaces que puedan ser utilizadas tanto en las plataformas móviles como en *Windows* manteniendo una misma lógica compartida [35] [36].

En la Figura 18, podemos apreciar un resumen de 2016[27] de la familia de entornos de desarrollo de *.NET* y los *App Models* que soportan.

[25] https://dotnet.microsoft.com/download

[26] Tal y como indica el propio artículo, "hasta junio de 2016", algunas de ellas han sido resueltas en la actualidad soportándose dichas características en versiones posteriores de *.NET Core.*

[27] Tiempo después de lo referenciado en [38], *.NET Core* ha ido recibiendo soporte para frameworks como *WPF* o *Windows Forms*, pero aun así la Figura 18 presenta un buen resumen de la familia *.NET.*

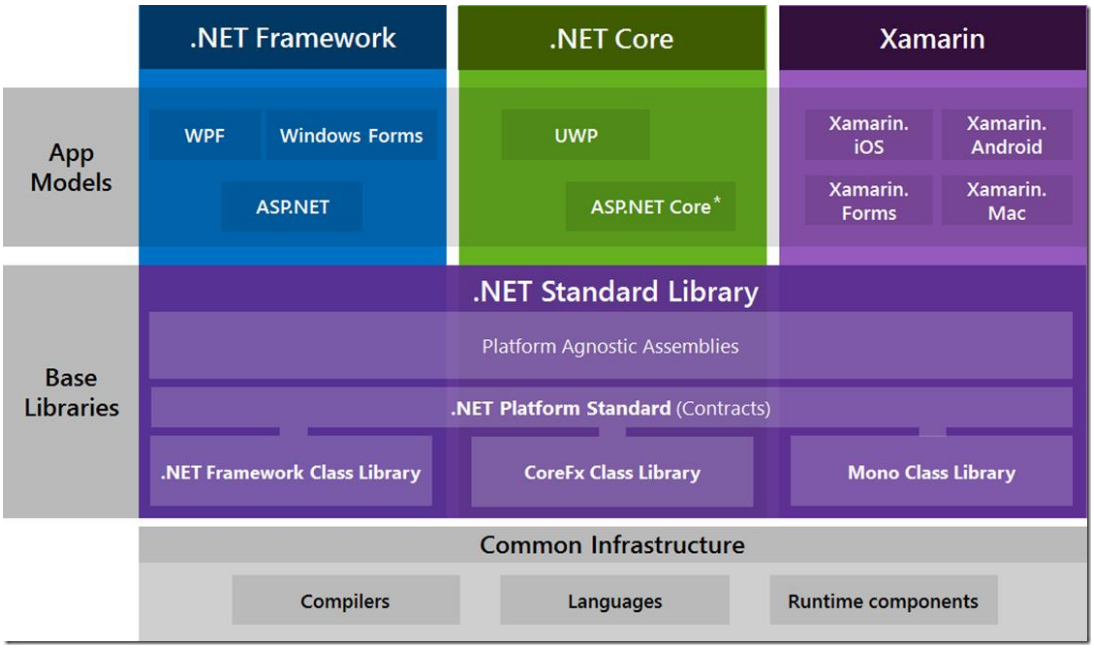

Figura 18. Resumen de la familia de frameworks de .NET [34].

3.2 Windows Presentation Foundation

El framework *Windows Presentation Foundation* (*WPF*), es un subsistema gráfico que ofrece un conjunto de herramientas y utilidades para el desarrollo de aplicaciones *.NET* para plataformas de escritorio. Una de las ventajas que ofrece, es que permite separar la interfaz de usuario de la lógica de negocio haciendo uso de archivos *XAML*[28] para ello.

Conviene mencionar especialmente el sistema de propiedades de *WPF*, que permite extender la funcionalidad de las propiedades de *C#*. Algunas de estas extensiones se basan en las denominadas *'DependencyProperties'*, que eliminan la necesidad de hacer uso del común campo privado de clase *'backing field'* para las propiedades y las proveen de nuevas funcionalidades, como permitir que puedan ser campos calculados. En otras palabras, que su valor dependa de otros datos de entrada como: preferencias de usuario, otras propiedades computadas en tiempo de ejecución, etc. [37].

Otra de las características de importancia de este *framework*, es la posibilidad de establecer *data bindings* [38]: las vistas o formularios de la interfaz enlazan campos de datos con los miembros de *C#* que proveen dichos datos. De esta forma ambas partes quedan sincronizadas, separando la vista de la lógica de negocio y permitiendo que se actualicen dependiendo del tipo de enlazado que se escoja: *one-time*, *one-way*, *two-way* o *one way to source* [39] [40].

Brevemente, comentar que los *behaviors* permiten definir comportamientos encapsulados comunes para que puedan ser reutilizados con diferentes elementos gráficos. Comúnmente, se emplean para definir comportamientos comunes para acciones del usuario como arrastrar, *bindear*, hacer foco, etc. [41] [42].

[28]Lenguaje de programación declarativa basado en XML para la especificación de interfaces de usuario [79].

3.3 OpenXML SDK y WordprocessingML

En primer lugar, conviene comentar el caso del *OpenXML SDK* [43] [44]. Este *SDK* se trata de un conjunto de clases que buscan simplificar y apoyar la manipulación de documentos que se adhieran al estándar *Office Open XML* [45] [46]. Es decir, *OpenXML* sería un estándar abierto para que se pueda trabajar con documentos, presentaciones y hojas de cálculo desde diferentes plataformas o suites ofimáticas. Gracias a ello, permite que se puedan editar documentos de igual forma tanto en, por ejemplo, *Word* como en *Writer*.

Por este último punto, se decidió que las Plantillas con las que trabajase este nuevo módulo de *Reports*, se basasen en este estándar. La ventaja de ello es principalmente la de no depender de una suite ofimática concreta, si no que estuviera abierto a cualquier editor de texto que soportase este estándar.

WordprocessingML sería la denominación en este estándar que reciben los documentos de texto [47]. En concreto, referencia el hecho de que estos documentos tienen una estructura concreta, basada en una serie de elementos tales como el Documento Principal, Comentarios, Pies de Página, Cabeceras, Definiciones de Estilo, etc. Como ya adelanta el propio nombre de *Open XML*, estos documentos disponen su estructura basándose en un esquema para *XML*, con algunos elementos comunes como *'p', 'r', 't'* (respectivamente, *paragraph*, *run* y *text*). En la Figura 19, se observa la estructura interna de los contenidos de un documento *WordprocessingML*; cómo se puede observar, en ella aparece un nodo texto, dentro de un nodo *run*, que a su vez se encontraría en un párrafo de lo que sería el cuerpo del documento.

```xml
XML

<?xml version="1.0" encoding="utf-8"?>
<w:document xmlns:w="https://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello, Word!</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document
```

Figura 19. Ejemplo del archivo interno 'document.xml' de un documento WordprocessingML [47].

En la Figura 20, se observa la estructura típica para un documento *WordprocessingML*.

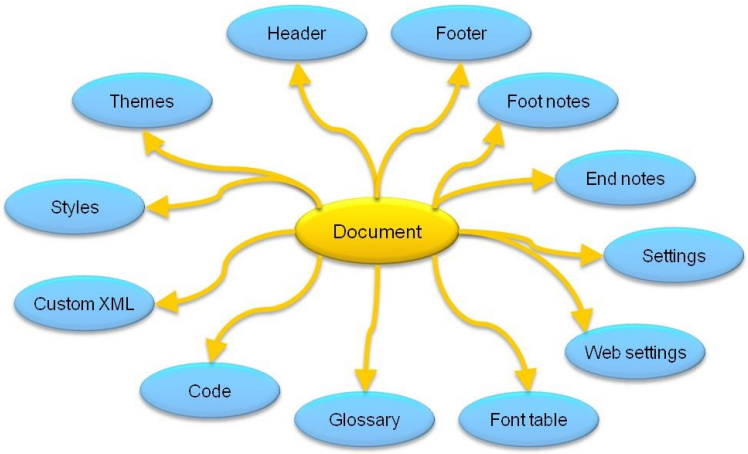

Figura 20. Ejemplo de la estructura típica para un documento WordprocessingML [47].

3.4 OpenXML PowerTools

Las *OpenXML PowerTools*[29] proveen de una serie de funcionalidades extra sobre el *OpenXML SDK*. Algunas de las extensiones y facilidades que ofrecen, relevantes en términos del Diseñador de Plantillas en desarrollo, serían tales como:

- Poblar contenido de una plantilla de documentos *DOCX* con datos desde un *XML*.
- Buscar y remplazar contenido en documentos *DOCX* usando expresiones regulares.
- Comparar dos documentos *DOCX*, generando un nuevo *DOCX* con el seguimiento de revisiones activado, y permitiendo generar una enumeración de las revisiones.

El primer punto, permite que *Reports Engine* sea capaz de, dada una Plantilla que contenga etiquetas o *tags* de las *OpenXML PowerTools*, cruzar un *XML* de datos (los *DTOs* de entrada de datos con los que se diseña una Plantilla de un *Report*) con dicha Plantilla para generar un *Report* propiamente dicho. Para realizar este cruzamiento entre la Plantilla y el *XML* de datos, se hace uso de los tags que se mencionaban anteriormente. Básicamente estos *tags* son cadenas especiales de texto que, siguiendo una estructura concreta y haciendo uso de *XPath*, indican como seleccionar los datos del *XML* de datos. El módulo de las *OpenXML PowerTools* encargado de realizar este procesamiento, sería el *DocumentAssembler*, que se explica en mayor detalle en las referencias [48] y [49]. Un ejemplo de estos *tags* sería el *'Content tag'*, que simplemente seleccionaría el valor de un nodo; su estructura sería *"<Content Select="./Name">"*, donde seleccionaría el valor del nodo *'Name'* a nivel raíz del *XML* de datos durante el procesamiento del documento *DOCX*.

Para el segundo punto, ofrece algunas facilidades a la hora de insertar los *tags* o hacer remplazos del contenido de la Plantilla, eliminando responsabilidades al equipo de desarrollo del presente Diseñador.

De cara al tercer y último punto, permite tener un mecanismo para que, en los tests con Plantillas, se pueda hacer una comprobación de similitud entre ellas y, en caso de que no sean exactamente iguales, se genere un documento *DOCX* con sus diferencias. Esto facilita mucho el desarrollo de aquellos tests que trabajan con comparaciones de Plantillas.

3.5 DevExpress WPF Rich Text Editor

DevExpress[30] es una empresa que ofrece librerías de controles de *UI* para editores de documentos tipo *WordprocessingML* y *SpreadsheetML* para el entorno *.NET*. En concreto, uno de los controles de interés para el presente trabajo es su editor de texto rico.

Este *Rich Text Editor* ofrece una serie de funcionalidades básicas de un editor de texto de tipo *WordprocessingML*, como son la edición de párrafos y texto del propiamente dicho texto rico, gestión de estilos, tablas, corrección ortográfica, operaciones de deshacer y rehacer, cabeceras y pie de páginas, campos de documento, tablas de contenidos y referencias, etc. [50]. Por supuesto, dado que no solo provee de la *Application Programming Interface (API)* para trabajar con ello, sino también la interfaz gráfica de usuario asociada (Figura 21), se permite acelerar y facilitar el desarrollo de aplicaciones que hagan uso de un editor de texto del estilo, como es el caso del Diseñador de Plantillas del proyecto.

[29] https://github.com/EricWhiteDev/Open-Xml-PowerTools
[30] https://www.devexpress.com/aboutus/ y https://www.devexpress.com/aboutus/

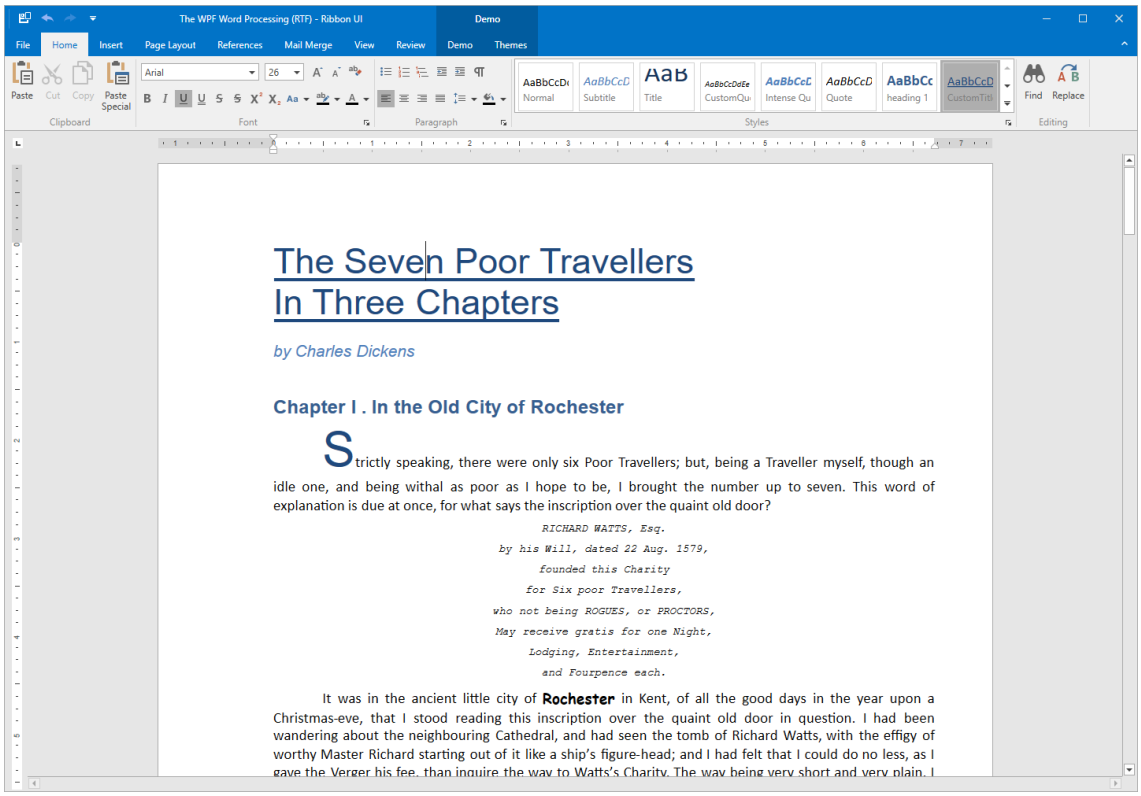

Figura 21. Ejemplo de la interfaz de usuario del Rich Text Editor control de DevExpress.

3.6 DSL Tools

Las *DSL Tools*[31] constituyen la propuesta de *Microsoft* para el diseño de un lenguaje específico de dominio y posterior generación de los artefactos modelados. Como su nombre indica, esta herramienta se basa en la definición de un *Domain Specific Language (DSL)* o Lenguaje Específico de Dominio: lenguajes creados con la finalidad específica de solucionar un problema para un dominio concreto. Este tipo de lenguajes, buscan eliminar complejidad innecesaria para permitir a un público que no sean desarrolladores resolver problemas de ese dominio concreto, con unas herramientas que cumplen con las necesidades exactas del mismo y además permitiendo acortar ciclos de desarrollo para dichos lugares del dominio donde puedan ser aplicables [51]. Así, en términos más técnicos y tal y como muestra la Figura 22, un *DSL* permite establecer una parte configurable que tiene alta variabilidad dentro de un contexto más invariable [52]. Por ejemplo, en el caso de un sistema genérico de gestión de billetes de avión, un *DSL* permitiría configurar el proceso de compra los mismos; de esta forma, existe una parte fija y genérica del sistema, así como una parte configurable que depende del cliente final.

[31] https://docs.microsoft.com/en-us/visualstudio/modeling/overview-of-domain-specific-language-tools?view=vs-2019

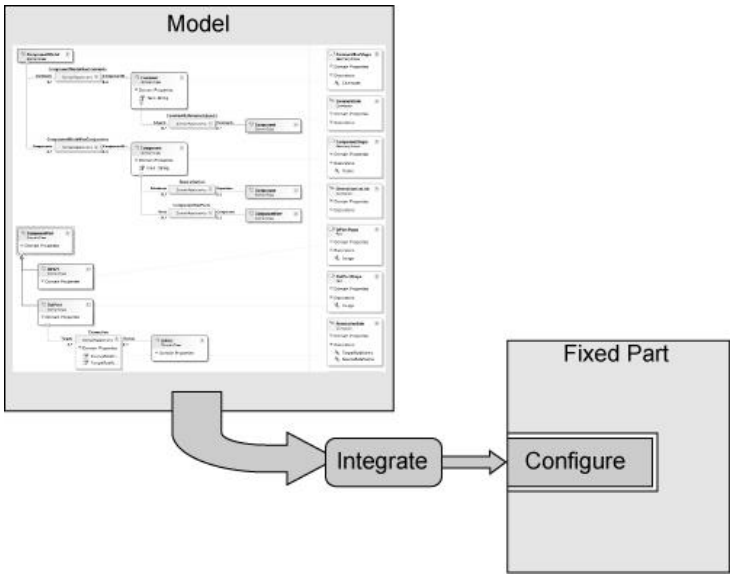

Figura 22. Cómo un DSL es una parte configurable en un contexto fijo [52].

El contexto de estos *DSL* es el Desarrollo Específico de Dominio o *Domain-Specific Development, "basados en la observación de que muchos problemas del desarrollo de software pueden ser más fácilmente resueltos mediante el uso de lenguajes diseñados con un propósito específico"* [52].

En concreto, las *DSL Tools* se engloban en lo que sería los *DSL* gráficos, lenguajes que hacen uso de una representación visual para modelar los diferentes conceptos y relaciones. Estas representaciones gráficas son de gran utilidad ya que permiten representar mejor las relaciones entre los diferentes conceptos o entidades, y se basan en hacer uso de formas y figuras concretas que representan de forma inequívoca estos conceptos. Sin embargo, es importante remarcar que los *DSL* gráficos no son únicamente diagramas, ya que estas figuras se encuentran ligadas a las entidades del modelo de dominio, y, por tanto, *"se están creando modelos que conceptualmente representan el sistema que estamos construyendo, con representaciones diagramáticas de los contenidos"* [52]. En la Figura 23, se puede observar el modelo de un proceso de compra, en el que aparecen diferentes figuras que representan conceptos y las relaciones entre ellos.

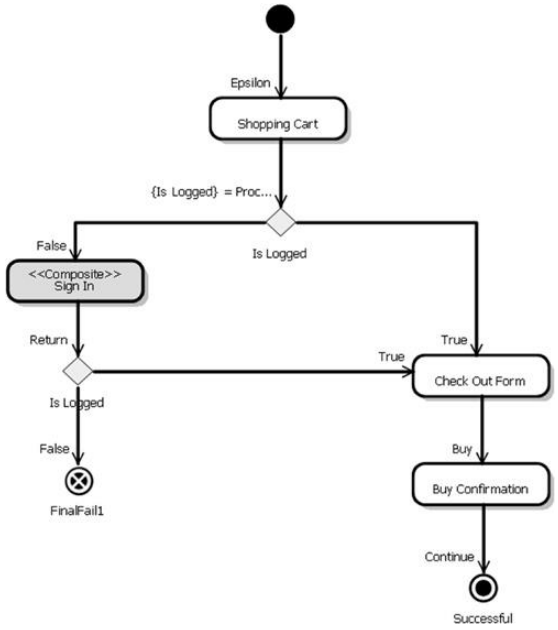

Figura 23. Ejemplo de un modelo diseñado mediante un DSL [52].

3.7 Microservicios

El Diseñador de Plantillas forma parte de la *UI* del microservicio de *Reports Designer*. Por ello, conviene hacer una breve introducción a los microservicos y arquitecturas basadas en ellos.

Citando a Sam Newman en *Building Microservices* [53], los microservicios son *"servicios independientemente publicables que son modelados alrededor de un dominio de negocio"*. En [54], se resumen como *"una forma de desarrollar una única aplicación como una suite de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose entre ellos mediante mecanismos simples, comúnmente mediante una API en HTTP".*

Entrando en mayor detalle, podríamos decir que estos servicios se encargan de cumplir con un único propósito, siendo por lo tanto de un tamaño muy reducido (de ahí el "micro" en microservicios)[32].

Otro punto de interés de los microservicios, y sobre a lo que hace referencia la parte de "independientemente publicables" en la anterior definición de esta arquitectura, es que lo que buscan es ser también independientemente desplegables. Esto significa que cualquier nueva versión de un microservicio se puede desplegar (instalar y configurar) sin necesidad de que otros microservicios que hagan uso de él tengan que volver a ser desplegados en conjunto, o incluso de que lleguen a enterarse de esta nueva versión del microservicio. Esto además tiene dos beneficios adicionales de interés[33]: uno de ellos es que permite que diferentes microservicios de un mismo producto se alojen en diferentes plataformas de diferente naturaleza y características (fiscal, rendimiento, etc.); el otro, es que ofrece flexibilidad de desarrollo al permitir que diferentes equipos se encarguen de diferentes microservicios y que trabajen de forma totalmente independiente aun cuando dichos microservicios interactúan entre sí[34].

[32] En [57] se comenta que "pequeño/reducido" es algo relativo y que depende del contexto y proyecto. En concreto ponen el ejemplo de "que pueda caber completamente en nuestra cabeza" sería un tamaño pequeño.

[33] Ambas se recogen en el capítulo 6 "*System Design and Operations*" de [57].

[34] Siempre y cuando los *endpoints* o comportamiento externo de dichos microservicios permanezca fijo según lo acordado.

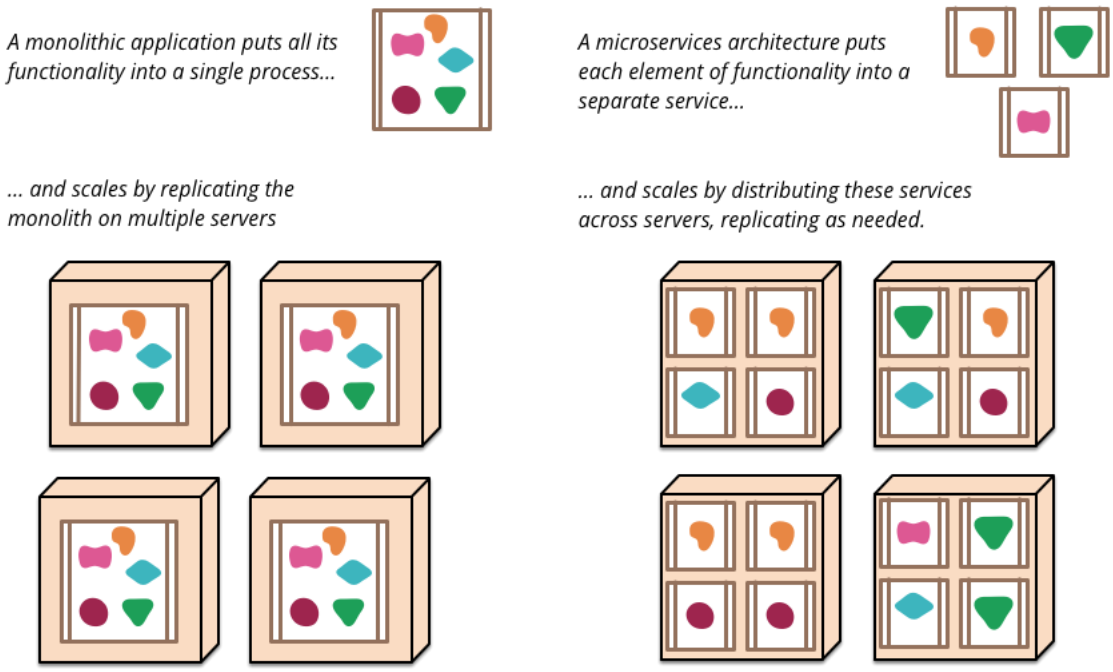

Figura 24. Comparativa de aplicaciones con arquitectura monolítica frente a una arquitectura con microservicios [54].

Poniendo en perspectiva una aplicación monolítica, encontramos que: un pequeño cambio obliga a redesplegarla al completo, y además solo puede escalar horizontalmente creándose más instancias de la propia aplicación como conjunto. Por el contrario, una aplicación basada en microservicios permite trabajar con una granularidad más fina: solo es necesario redesplegar aquellos microservicios en los que se hayan hecho cambios (de forma similar a cuando se estructura una aplicación en diferentes ensamblados o paquetes, permitiendo *builds* incrementales[35]), así como escalar e invertir recursos en aquellos microservicios que así lo requieran específicamente. Sin embargo, por ejemplo, una aplicación monolítica es mucho más simple y por lo tanto más fácil de desarrollar y depurar en comparación; no tiene el sobrecoste de las comunicaciones entre procesos o invocaciones remotas; y además no requiere de una infraestructura tan compleja como si lo hacen los microservicios que además necesitan de un esfuerzo extra en el diseño, despliegue y operación (por ejemplo, como plantear el descubrimiento de los microservicios, si existe un orden de despliegue óptimo para la resolución de dependencias y ejecutarlo, casuísticas asociadas a la operación y comunicaciones distribuidas, etc.) [55].

[35] De igual forma, *builds* que permiten reutilizar los ensamblados que no han recibido cambios, y así únicamente necesitando recompilar los que si los hayan recibido, mejorando por tanto el tiempo de compilación.

4. Propuesta de módulo Diseñador de *Plantilla*s

En este apartado se expondrá finalmente cual es la propuesta de módulo Diseñador de Plantillas. En primer se explicará de la metodología de trabajo, pasando a comentar cuáles han sido las mejoras implementadas, así como su especificación de requisitos, diseño y arquitectura en las que fuese de interés; además se comentarán las pruebas que se hayan realizado para verificar la funcionalidad que implementan las mejoras. En primera instancia se presentarán los aspectos generales de estos apartados que sean compartidas para todas las mejoras, para luego pasar a comentar dichos apartados al respecto de cada una en específico, si procede. Por último, se expondrán los resultados obtenidos del desarrollo realizado.

4.1 Metodología de trabajo

En el presente apartado se expondrán los puntos más relevantes sobre la parte metodológica del proyecto. En primer lugar, se comentarán brevemente las fases por la que ha pasado el proyecto, pasando por mostrar el plan de trabajo que se ha seguido y terminando con las menciones a las entregas de funcionalidades que se iban haciendo.

Para el desarrollo de las mejoras del diseñador de *Plantilla*s, se adoptó el flujo de trabajo y cultura de la empresa hasta el momento, lo cual se explicará a continuación:

Al respecto de la especificación y diseño, si la funcionalidad a implementar era de gran importancia o la interacción del usuario con la misma no era clara, se realizaba una especificación en detalle. En caso contrario, la especificación de la mejora a implementar se comentaba con el propietario del producto e interesados del equipo de desarrollo, acordando los puntos esenciales y, de manera informal, como debería comportarse. Durante la implementación de dicha funcionalidad, dichas partes interesadas (incluyendo al propio propietario del producto) iban revisando constantemente el progreso y resultados parciales de esta, de manera de que si había algo que se consideraba que necesitaba modificarse no tardaba mucho en exponerse y actuar en consecuencia: redefinir el alcance, variar el comportamiento del sistema en alguna casuística, etc. En resumen, el grado de detalle de la preparación de las mejoras variaba dependiendo de la complejidad, importancia y claridad de la funcionalidad en cuestión.

En el caso del diseño y arquitectura, estos aspectos iban siendo abordados durante el desarrollo de las mejoras; principalmente, esto se debe a que las mejoras no solían imponer grandes cambios en la arquitectura ya existente y solían adecuarse a la misma. Los casos en los que sí se requería de un diseño más profundo de antemano, eran aquellos en los que la dificultad técnica era alta o se debían de tener en cuenta ciertos detalles de implementación. Sin embargo, en la mayoría de los casos el diseño se abordaba después de tener una primera implementación funcional, evaluando técnicas de refactorización como: extraer de lógica común o redundante a métodos o clases, añadir interfaces para invertir dependencias, etc. Esto se vio reforzado cuando se empezó a aplicar *Test Driven Development (TDD)* [56], donde primero se escribe un test que falla y especifica el comportamiento que se quiere introducir, después se añade el mínimo código de producción que hace que pase el test, y una vez se tiene una primera implementación funcional, entonces se pasa a refactorizar y mejorar el diseño de la misma. Uno de los muchos beneficios que ha aportado esta técnica, es la de estar siempre cubierto en las refactorizaciones que hacemos: siempre tenemos uno o varios test que verifican que el comportamiento del sistema sigue siendo el que se espera. Otro de los beneficios que ofrece el *TDD*, es el de poder ajustar a placer la granularidad de los cambios que se realizan sobre el sistema, permitiendo así poder avanzar de forma más segura en contextos de alta incertidumbre o dificultad técnica e ir más rápidos cuando la incertidumbre sea baja.

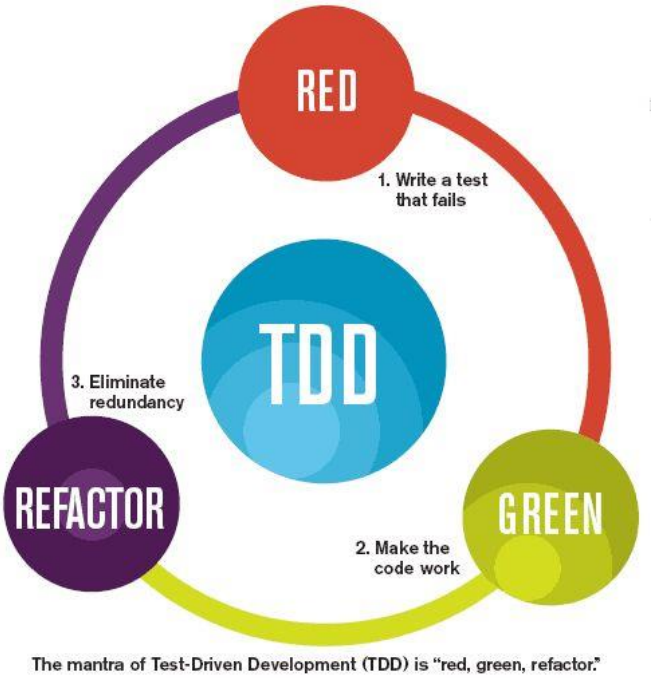

Figura 25. Bucle de desarrollo haciendo uso de Test Driven Development [57].

Por esto último que se comenta, y por propia naturaleza del *TDD*, las pruebas unitarias y de integración sobre algunas de las mejoras (a partir del punto en el que se empezó a aplicar dicha técnica) cubrían totalmente el comportamiento de la nueva funcionalidad a implementar.

4.1.1 Fases

El inicio del proyecto estuvo marcado por una primera fase de formación e introducción al proyecto, donde principalmente se llevaron a cabo mejoras menores sobre el Diseñador de Plantillas. Durante este periodo, se requería de un alto grado de supervisión tanto en términos de revisiones de código como en apoyo a la propia implementación de la mayoría de los cambios en el código.

Una vez se fue asentando el conocimiento sobre el proyecto y metodología de trabajo, se fueron abordando mejoras algo más complejas, intercalándolas con otras más simples. Esto vino acompañado de un mayor grado de independencia con respecto a los mentores en el proyecto gracias a los conocimientos y experiencia adquiridos, pero dada la complejidad y continuo descubrimiento de nuevos aspectos del contexto del propio proyecto, se seguía necesitando de gran ayuda por la parte de susodichos revisores. Además, durante esta fase se aprovechó para mejorar la documentación interna del equipo de desarrollo, añadiendo nuevas entradas a la wiki interna y mejorando las ya existentes.

Cerca del final del periodo de este proyecto, la atención se puso sobre todo en la preparación del segundo hito del Centro para el Desarrollo Tecnológico Industrial *(CDTI)*[36]. Para este hito, se planificaron dos tareas principales:

- La primera era relativa a la generación de *Reports* contextualizados. Esto se abordó por otro integrante del equipo encargado de *Reports*, por lo que no se ha considerado en este trabajo.
- La otra tarea sí que fue abordada por nosotros, y además ocupó la totalidad de la dedicación al proyecto durante este periodo. Esta mejora fue la de localización y traducción de Plantillas, tal y como ya se ha especificado anteriormente en el apartado correspondiente. Fue justo con esta mejora con la que se empezó a experimentar aplicando *TDD,* y donde como ya se ha explicado anteriormente, el resultado fue bastante satisfactorio.

4.1.2 Plan de trabajo

El plan de trabajo establecido al principio del proyecto estuvo marcado por el ritmo que permitían las prácticas en empresa. Si bien el trabajo principal desempeñado en dichas prácticas fue la mejora del Diseñador de Plantillas de *Reports* propiamente dicho, también existían otras tareas y objetivos a cumplir que competían con el tiempo de dedicación. Es por ello por lo que desde un inicio se acordó que el plan de trabajo fuera flexible, permitiendo que se invirtiese la mayor parte del esfuerzo en el desarrollo de las mejoras requeridas, pero también que dichas mejoras se pospusieran ante otras tareas que requiriesen de atención. Dicha flexibilización consistía en que se fueran planteando las diferentes mejoras en forma de unidades de trabajo[37], y siendo estas recogidas en el *backlog*[38]. Así, conforme se iba terminando una unidad de trabajo, se cogía una nueva desde el ya mencionado *backlog* de *Reports*.

4.1.3 Entregas

Ya que el producto en desarrollo todavía no se encontraba en su fase de explotación, ni los clientes ni el resto de los interesados en el mismo requerían de entrega alguna. Aun así, durante todo el trascurso del proyecto se intentó que, siguiendo el flujo de trabajo y cultura de la empresa, se fueran acortando los ciclos de desarrollo al entregar las nuevas funcionalidades.

Esto básicamente se resume en hacer pequeños incrementos que aporten las unidades funcionales mínimas para las diferentes características que se piden del Diseñador de Plantillas. Así, se buscaba que las *pull requests (PR)*, utilizadas como medio de comunicación y revisión de código [58], tuvieran los menores cambios necesarios para que su integración en la rama principal tuviera sentido. De esta forma, al reducir el tiempo que se tardan en hacer visible los cambios al resto del equipo, se acortaban también los ciclos de desarrollo y por tanto el *feedback* de los diseñadores e interesados del equipo de desarrollo era más constante y valioso, ya que permitía hacer los cambios necesarios lo antes posibles. Por ello, estos cambios solían ser pocos al reevaluar el producto de forma tan constante y granular; cuando se daba el caso de que sí se requería algún cambio, solía ser uno de gran relevancia para la característica en cuestión y que se había escapado a la especificación inicial de los requisitos.

---

[36] https://www.cdti.es/index.asp?MP=100&MS=898&MN=1&r=1920*1080

[37] Contenedores genéricos de tareas, funcionalidades o cualquier unidad de esfuerzo a llevar a cabo. Más detalle en [82].

[38] Lista priorizada de los diferentes ítems (unidades de trabajo, tareas, etc.) no acabadas asociadas a un producto o línea de trabajo. Más detalle en [81].

4.2 Mejoras realizadas

A continuación, pasamos a exponer en detalle las diferentes mejoras abordadas durante el desarrollo de este proyecto. Como ya se ha comentado al inicio del capítulo ("4. Propuesta de módulo Diseñador de *Plantilla*s"), ahora se hablará de la especificación de requisitos, diseño y arquitectura, desafíos de programación y pruebas al respecto de cada mejora en específico.

Cabe mencionar que, recordando lo que ya se ha comentado anteriormente, si bien se hacía una especificación de requisitos inicial, tanto la propia especificación como diseño y arquitectura se iba reevaluando de forma constante a lo largo de la implementación. Por ello, en la mayoría de los casos, en dichos apartados concretos de las diferentes mejoras se irá plasmando esa evolución.

M1: Soportar la inserción y procesado de Secciones Repetibles sobre colecciones de *DTOs* en diferentes formatos

Esta mejora trata sobre el soporte a la inserción de Secciones Repetibles desde el Diseñador de Plantillas, de manera que, para los Listados, se puedan diseñar unas regiones con contenido que se repetirá para los diferentes elementos de la colección de datos de entrada. Estas regiones podrán insertarse con dos formatos distintos: en formato card, donde el Valor de cada Campo de la entidad se inserta a continuación de su nombre a mostrar (su *"Display Name")*; otro correspondería al formato tabla, donde el *Display Name* del Campo de la entidad, que es independiente del Valor que este tenga, se insertará en una cabecera común para todas las ocurrencias de la Sección Repetible, y el Valor de dicho Campo se inserta en la región que se va a repetir, propiamente dicha.

El caso de una de estas colecciones de *DTOs*, podría ser el de un *Report* que recoge los nombres y edades de los Residentes de una de las plantas o sedes de la organización. Así, del *DTO* de entrada de datos para el *Report* sería una colección de *DTOs* de Residentes, con los Campos Nombre y Edad para cada uno de ellos. Por lo tanto, en la Plantilla de dicho *Report* existiría una Sección Repetible que especifique el diseño para los Campos de cada uno de los Residentes.

Cabe comentar que esta mejora se implementó en dos *PRs* distintas: en la primera se implementó toda la lógica de inserción y procesado de Secciones Repetibles para el caso del formato *card;* la segunda hizo uso de la lógica ya implementada en la primera para simplemente dar soporte al formato tabla y diálogo con el que el usuario puede elegir qué formato prefiere a la hora de insertar cada colección de *DTOs* de los datos de entrada.

M1: Especificación de requisitos

Como ya se exponía anteriormente, las Secciones Repetibles se insertan para colecciones de *DTOs*, ya que por definición lo que especifican es el diseño que va a tener cada una de las ocurrencias de los elementos de la colección. De esta forma una Sección Repetible permite, y da soporte, al diseño de *Reports* con colecciones de *DTOs* (lo que serían Listados).

Si bien la descripción que se plantea al inicio del apartado "M1: Soportar la inserción y procesado de Secciones Repetibles sobre colecciones de *DTOs* en diferentes formatos" fue la que se propuso como especificación inicial de la mejora, podríamos profundizar un poco en los detalles de la misma con la explicación de lo que sería el "camino feliz" o "ideal" de la inserción de Secciones Repetibles.

En el caso de que el *DTO* de datos de entrada sea una colección o de que alguno de sus Campos sea o contenga recursivamente una colección de *DTOs*, si el usuario que está diseñando la Plantilla hace doble click o arrastra dicha colección, se entiende que lo que quiere es insertarlo en la Plantilla, al igual que con el resto de Campos del *DTO* de datos de entrada. Aquí en una primera implementación en la que sólo se soportaba el formato *card* y, donde por lo tanto no había elección posible, se procedería a insertar la colección en formato *card.* En el caso de que, mientras se itera sobre los Campos del *DTO* en cuestión, se encontrase a su vez otra colección de *DTOs*,

se procedería a insertar recursivamente una Sección Repetible anidada con el mismo formato que su Sección Repetible padre; esto se repite hasta llegar a los nodos finales de la estructura arbórea del *DTO* de entrada de datos. Una vez se acabó dando soporte al formato tabla en una mejora posterior, cuando se detectase que el usuario quería insertar una colección de *DTOs*, se le mostraba un diálogo preguntando si lo deseaba hacer en formato tabla o formato *card* para actuar en consecuencia.

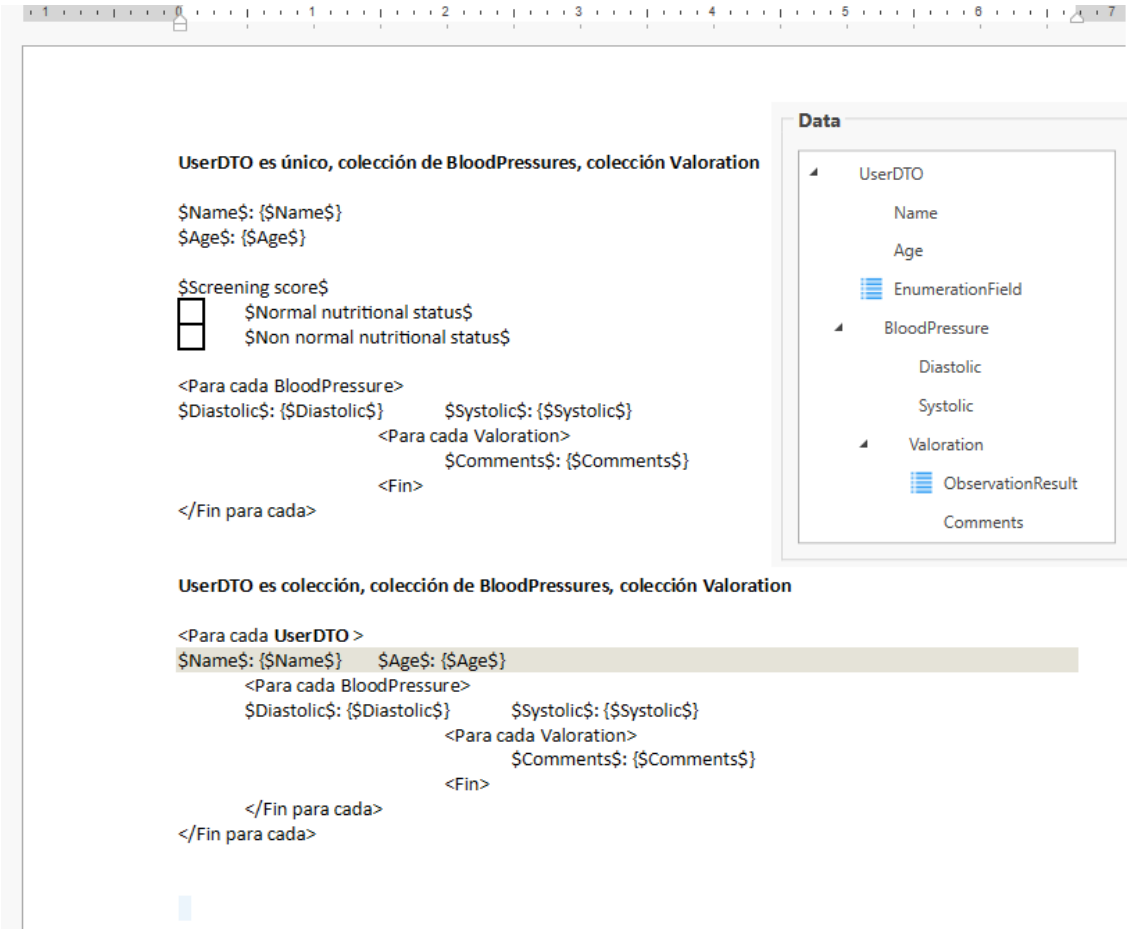

Figura 26. Primer diseño conceptual de la especificación sobre cómo debería verse una Sección Repetible en formato card. En la parte derecha puede observarse el DTO de datos de entrada.

En la Figura 26 se puede observar una de las primeras especificaciones que se hicieron sobre cómo el usuario diseñando la Plantilla debería ver una Sección Repetible en formato *card*. En la primera parte, puede observarse el caso en el que el *DTO* de datos de entrada (llamado *"UserDTO"*) es único, y sólo su Campo "*Blood Pressures"* sería una colección. A su vez, se encontraría que uno de los Campos de una *Blood Pressure*, *Valoration*, sería también a su vez una colección, por lo que se insertaría como una Sección Repetible anidada; En la segunda parte, puede verse la casuística en la que el propio *DTO* de datos de entrada fuera también una colección, por lo que se insertaría como una Sección Repetible. Tal y como se ha comentado para el caso anterior, su Campo *Blood Pressures* y a su vez el Campo *Valoration* serían insertados como Secciones Repetibles anidadas.

Un ejemplo de cómo sería aproximadamente un *Report* generado con Secciones Repetibles anidadas a varios niveles, sería el que se puede apreciar en la Figura 27. Decimos aproximadamente, ya que los *Display Names* de los Campos aparecen con los *tags* delimitadores de Elementos, y no han sido remplazados por los valores reales. Sin embargo, es suficiente para mostrar las Secciones Repetibles anidadas tal y como se pretende.

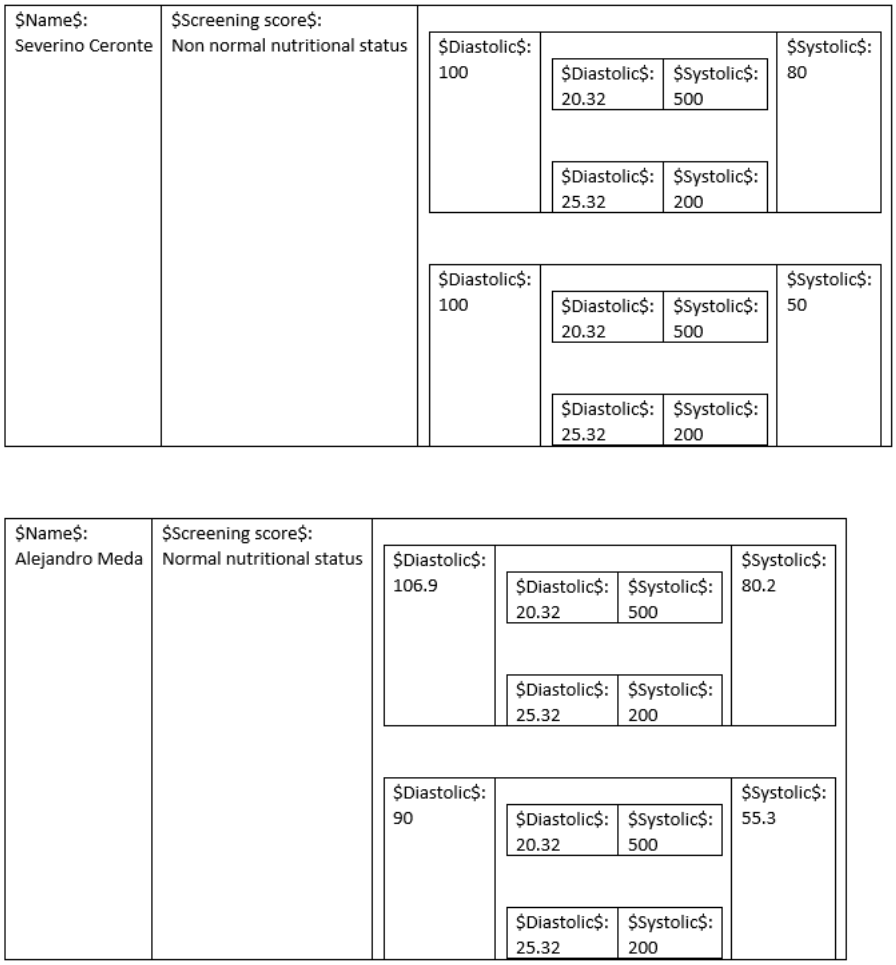

Figura 27. Ejemplo de Report generado con Secciones Repetibles en formato card a varios niveles de anidamiento.

Mencionar además que el diseño conceptual inicial que se muestra en la Figura 26, no muestra las Secciones Repetibles implementadas mediante tablas del documento como se estaba comentando. Un ejemplo de una especificación posterior en la que ya se ve muy cercana a la implementación real, sería el que se puede apreciar en la Figura 28. Mencionar que, el segundo formato que se nombra como "Sección Repetible", sería el que se ha acabado denominando como "card".

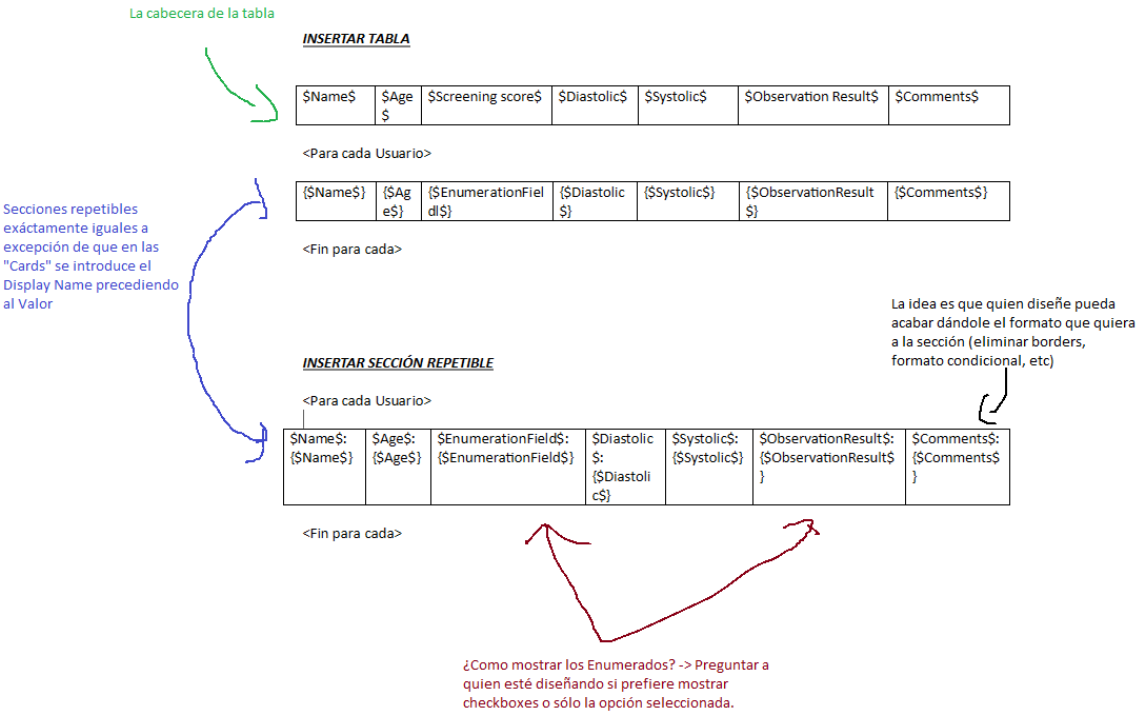

Figura 28. Boceto con anotaciones sobre los dos formatos de Secciones Repetibles.

Más adelante en el desarrollo, y con respecto de los Campos de tipo Enumerado en las Secciones Repetibles, se acordó que se insertase cada una de las diferentes opciones de la enumeración de forma independiente, y no con los *n* Valores con *checkboxes* (tal y como aparece para el Campo "*Screening Score"* en el primer caso de la Figura 26). Esto era necesario para poder soportar el hecho de que se mostrase el Valor del Enumerado como su valor en cadena de caracteres, y no como simplemente el valor interno de la enumeración (0, 1, 2, etc.).

```
<#<Repeat Select="./88944870-af32-4638-9919-32d98cc536ce" Optional="true" />#>
$Nested Domain Enumeration Display Name$:
<#<Conditional Select="./60382db2-b0bb-4020-9e18-b1dc91df7de6" Match="1" Optional="true" />#>
33bca634-fe33-4a08-8d83-caa89619fb05
<#<EndConditional />#>
<#<Conditional Select="./60382db2-b0bb-4020-9e18-b1dc91df7de6" Match="2" Optional="true" />#>
d5ea85c5-12b5-41d4-8569-91efb00753ca
<#<EndConditional />#>

<#<EndRepeat />#>
```

Figura 29. Ejemplo de un Campo de tipo Enumerado dentro de una Sección Repetible en una Plantilla guardada en el Diseñador de Plantillas.

En la Figura 29 podemos observar esta casuística para una Sección Repetible de un *DTO* con un único Campo, que sería de tipo Enumeración. Esta Enumeración tiene dos opciones, la primera correspondería a la que tendría en su Valor a mostrar el identificador "33bca634-…", y la otra correspondería con el identificador "d5ea85c5-…". Como explicación más en detalle, decir que ambas opciones seleccionan el mismo Campo del *DTO*, ya que podemos apreciar como el *XPath* de sus respectivos *Select* son equivalentes; sería el atributo *"Match"* el que especificaría que corresponden a los Valores 1 y 2 respectivamente. En pocas palabras, lo que esto viene a decir es que: si un Campo de tipo Enumerado toma el Valor 1, se mostrará su texto (*Display Name)* asociado; si toma el Valor 2, su correspondiente equivalente en texto, etc. Por último, mencionar que lo que se ve en esta Figura 29 se mantiene oculto al usuario del Diseñador en todo momento, y sería el contenido real del documento y que el motor de *Reports (*microservicio

Reports Engine) podría entender para generar finalmente el *Report* como tal. En la Figura 30 podríamos apreciar como sí vería el usuario un Campo de tipo Enumerado (Campos con *Display Name "Screening Score"* y *"Observation Result"*).

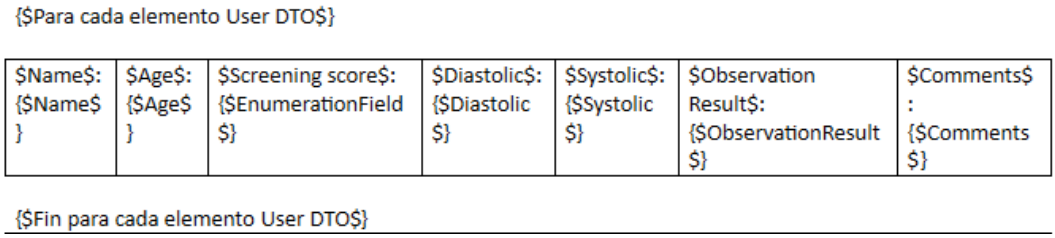

Figura 30. Ejemplo de cómo vería un usuario del Diseñador de Plantillas un Campo de tipo Enumerado (Screening Score u Observation Result) en una Sección Repetible.

Otra cuestión que se tuvo que discutir durante la implementación del soporte a insertar Secciones Repetibles en formato tabla, fue la de cómo proceder cuando se encontrara la necesidad de insertar Secciones Repetibles anidadas a la hora de insertar la cabecera de la tabla. Tras una breve discusión con los analistas, se determinó que lo más conveniente era simplemente seguir el mismo comportamiento de la inserción de Secciones Repetibles en formato *card* e insertar los *Display Names* en tablas anidadas. Esto se puede apreciar en la Figura 31, donde aparece el caso de una Sección Repetible en formato tabla con una Sección Repetible anidada en la que aparece la cabecera con los *Display Names* correspondientes en una tabla anidada.

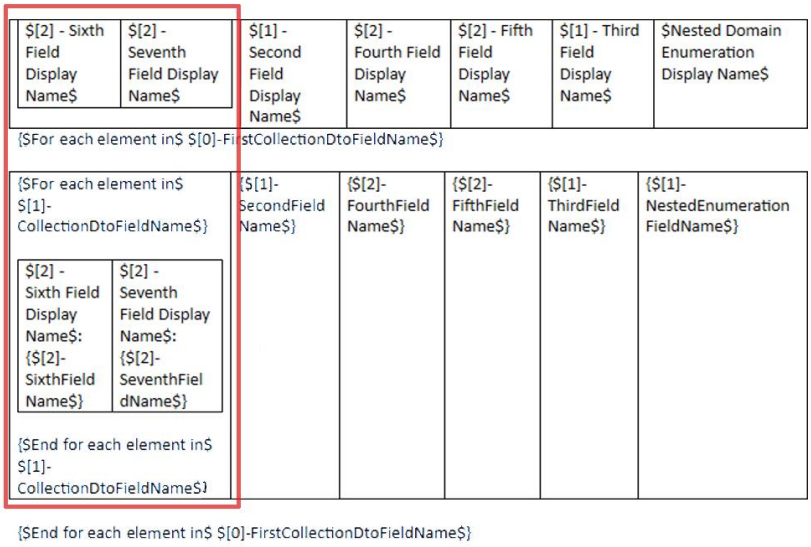

Figura 31. Ejemplo de Sección Repetible en formato tabla, donde se marca el caso de una Sección Repetible anidada.

Por último, mencionar que en un inicio se empezó hablando de implementar un asistente para la inserción de estas Secciones Repetibles con sus diferentes formatos (Figura 32). Este asistente ofrecería principalmente la opción de seleccionar entre los dos formatos ya comentados, como tabla o como *card;* además, también daría la opción de seleccionar la distribución de los Campos en la tabla de la Sección Repetible: número de filas total o número máximo de columnas por fila. Adicionalmente, se propuso dar opción a elegir el formato de los Campos de tipo Enumerado: mostrando cada opción por separado con *checkboxes* o mostrando la Enumeración como el resto de Campos y que a la hora de la generación únicamente apareciese el valor de la Enumeración. Al final, esta especificación quedó aplazada ya que simplemente con dar soporte a la selección de formato mediante un diálogo (Figura 33) fue suficiente, y no se continuó con el asistente en cuestión.

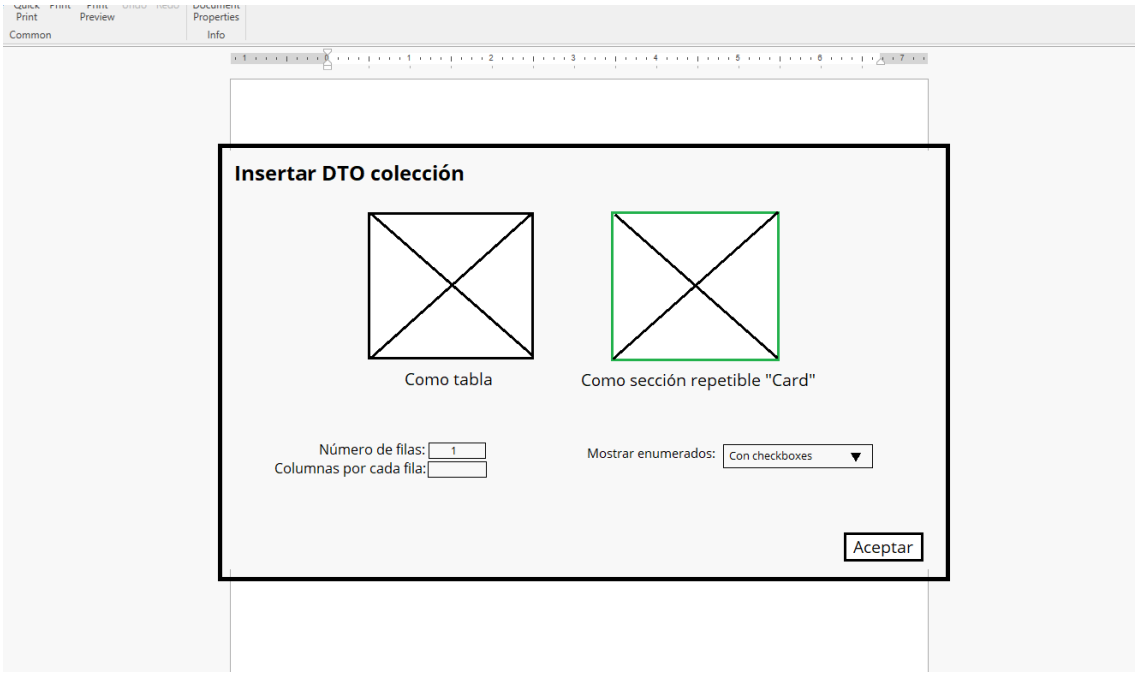

Figura 32. Mockup del asistente de inserción de colección de DTOs.

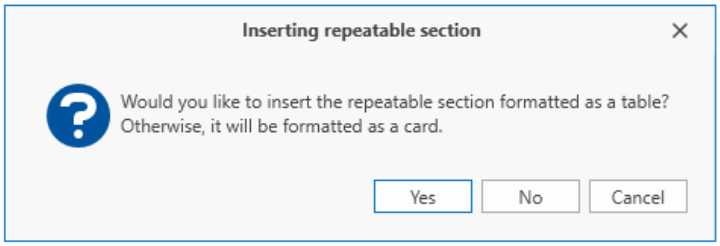

Figura 33. Diálogo que aparece al hacer click sobre un DTO colección que pregunta sobre en cuál formato insertar la Sección Repetible.

M1: Diseño

En primer lugar, se pensó que, la mejor forma de implementar estas Secciones Repetibles dando las mayores facilidades al diseño, era la de hacer uso de tablas. Esta decisión se fundamentaba en el hecho de que las tablas ofrecen una buena base para maquetar el contenido de las diferentes celdas, ya que permiten hacer uso de los separadores entre ellas (tanto para filas como para columnas) para organizar su distribución de forma muy directa y visual. Además, la organización en diferentes filas de la información concilia bastante bien con el hecho de que cada fila corresponde a cada ocurrencia de la colección y cada columna a cada Campo de la Entidad. Por otro lado, el comportamiento de que las tablas se puedan fusionar permite que a la hora de generar el *Report*, cada fila (que a su vez es una tabla en si misma) se fusione con sus adyacentes, generando así una única tabla para la Sección Repetible. Al respecto de esto último, mencionar que para que las tablas en Secciones Repetibles se fusionen correctamente tras la generación de un *Report*, es necesario que no haya separación entre los *tags* de control. En último lugar, comentar que el soporte en la *API* de *OpenXML* para trabajar con tablas también facilita la inserción de las Secciones Repetibles en el documento de la Plantilla.

Así, como primer acercamiento a la implementación de la selección de Campos del *DTO* de datos y mecanismo encargado de "repetir la sección marcada para cada ocurrencia", se pensó en el uso del *tag 'Table'* de las *OpenXML Power Tools*. Básicamente, este *tag* busca en el documento una tabla justo a continuación de sí misma, donde seleccionará los nodos indicados en ella relativos al nodo seleccionado en el *Select* del *tag*. Para cada ocurrencia del nodo seleccionado en

el *tag*, se añadirá una fila a la tabla del documento que contendrá los valores de esa ocurrencia[39]. Un ejemplo del *tag Table* sería el que se puede observar en la Figura 34.

Figura 34. Ejemplo de una tabla marcada con el tag 'Table' de las OpenXML Power Tools.

Sin embargo, la mayor problemática que ofrecía este *tag* era que era muy poco flexible en cuanto a posibilidades de formateo de la Sección Repetible, ya que obligaba a que la región a repetir siempre estuviera dentro de una tabla. Además, lo que hizo desestimar esta solución de implementación para las Secciones Repetibles fue que no permitía el uso de *tags* condicionales en las mismas: a la hora de que las *OpenXML PowerTools* procesasen un documento con esta casuística, generaba un fallo de formato y no se seleccionaba el valor correspondiente. Dado que, como se explicaba anteriormente, se requería que la inserción de Campos de tipo Enumerado en Secciones Repetibles se hiciera mostrando el texto asociado al valor del Campo, estábamos obligados a que esta inserción se hiciera mediante *tags* condicionales. Por ello, finalmente se acabó haciendo uso del *tag 'Repeat'*, que básicamente se define como "una sección la cual se va a repetir para cada ocurrencia del nodo seleccionado". Este *tag* sería una especie de versión más genérica del *Table*, no estando limitados a tablas (o pudiendo hacer uso de no una única tabla sino de varias), dando mayor flexibilidad en cuanto al diseño y, como punto más relevante, permitiendo el uso de *tags* condicionales[40].

M1: Desafíos de programación

Una de las primeras problemáticas que se encontraron al respecto de esta mejora, fue la forma de identificar las Secciones Repetibles del documento. Como se estaban implementado como tablas, una primera idea fue la de hacer uso de algún identificador interno de las mismas o incluso la posibilidad de añadirle metadatos como un nombre, pero se descartó rápidamente dado que no parecía haber un soporte para ello en la *API* de *OpenXML SDK*.

Un aspecto para tener en cuenta sobre el mostrado al usuario de los Campos en la Plantilla, es que se requiere mostrar el texto asociado de cada Elemento y no su identificador. Para ello, el mecanismo utilizado hasta el momento es el de, a la hora de insertar un Campo, guardar su correspondiente texto asociado junto a su identificador; esto se acaba serializando en el documento, de forma que, al abrir una Plantilla, se pueden remplazar los identificadores encontrados por su texto asociado. La mención a este mecanismo es relevante ya que, durante el desarrollo de la presente mejora, se encontró la problemática de que al aplicar dicho mecanismo a los Campos de tipo Enumerado en la Sección Repetible (insertados con la casuística que se comentaba en "M1: Especificación de requisitos"), ocurría un fallo dado que se intentaba buscar el texto asociado a los identificadores de las diferentes opciones cuando estos no habían sido persistidos en el documento. La razón de ello es, tal y como ya se comentaba anteriormente, que el usuario del Diseñador de Plantillas nunca llega a ver las diferentes opciones de la Enumeración en la Plantilla (de nuevo, lo mostrado en la Figura 29 estaría oculto al usuario, que solo vería algo parecido a la Figura 30), y por lo tanto no es necesario que éstas se remplacen. Aunque estos identificadores no fueran visibles al usuario, estaban presente de igual forma en el documento, lo que hacía que el mecanismo de remplazo fallara al intentar hacer su remplazo. Tras investigar durante un tiempo, se acabó encontrando esta razón del fallo, y se decidió que dichos identificadores no se insertaran con los tags que propiamente delimitan identificadores. Para

[39] Para mayor detalle, consultar la referencia [83] en el minuto 8:45 hasta el 12:23.
[40] En la referencia [83], desde el minuto 12:23 hasta el 15:00 se hace una breve explicación del *tag*.

permitir que el *backend* del Diseñador pueda reconocer estos identificadores, se hace un preprocesamiento en el que se buscan identificadores que se encuentren solos en una celda de una tabla (recordemos, que las Secciones Repetibles se han implementado haciendo uso de tablas); una vez encontrado, simplemente se envuelve el identificador con estos delimitadores que el *backend* entiende.

Otra problemática que se encontró durante el desarrollo de esta mejora fue el caso de que el editor de *RichEdit* de *DevExpress* guardaba los *.DOCX* con un estilo distinto al del estándar de *OpenXML*. Concretamente, ocurría un fallo al validar el documento guardado, donde para las propiedades de los párrafos [59] se colocaba el nodo *'BiDi'* [60][41] fuera del orden correcto. Tras investigar y abrir *issue* [61], se descubrió que el nodo de las propiedades de los párrafos, *'rPr'*, se define en el estándar como una secuencia, donde por lo tanto el orden de sus nodos hijos es relevante. Para evitar este error de validación, simplemente se decidió añadirlo a la lista de errores esperados del validador, de forma que no avisara del mismo.

M1: Pruebas

Para esta mejora, las pruebas que se llevaron a cabo fueron principalmente de índole manual. Así, conforme se iba dando soporte incrementalmente a las Secciones Repetibles, se iba probando a diseñar Plantillas en el propio Diseñador con esta nueva casuística y pasarlas por el *Engine* para que finalmente se generase el *Report*. Así, se observaba este *Report* generado en *Word* y *LibreOffice Writer* para comprobar que todo se visualizase como se esperaba. En algunos casos, también se analizaba el documento generado a nivel interno y sus diferentes partes; esto se hacía cambiando el formato del *.DOCX* por *.ZIP* e inspeccionando los archivos que en ese caso procediera, ya que un *.DOCX* no deja de ser un conjunto de archivos *XML* [62] [63].

M2: Ocultar las operaciones intermedias con los comandos de deshacer y rehacer para inserciones de Campos de la Plantilla

Esta mejora trata de mejorar el comportamiento de los comandos de "deshacer" y "rehacer" del control de *DevExpress*, de forma que las operaciones intermedias de las inserciones de Campos queden ocultas al usuario del Diseñador. En otras palabras, que las inserciones de Campos en la Plantilla se comporten como transacciones.

M2: Especificación de requisitos

La motivación a esta mejora es que a la hora de insertar Campos del *DTO* de datos de entrada en la Plantilla, se hacen una serie de modificaciones para que el usuario del Diseñador vea el Campo de una forma más *user-friendly;* estas modificaciones, se denominan operaciones intermedias. Algunas de ellas serían ocultar el texto de '*DOCVARIABLE'* aparece al insertar un campo del documento[42] o la de mostrar el Código del campo del documento en vez de su Valor[43]. El comportamiento de los comandos "deshacer" y "rehacer" antes de implementar la mejora, era tal y como se indica a continuación:

Modificaciones anteriores → Inserción del Campo → Ocultar el DOCVARIABLE → Modificaciones posteriores.

Cuando realmente lo que se requeriría es que fuera tal que:

---

[41] Este nodo especifica si el párrafo fuera de izquierda a derecha o viceversa.

[42] El Control de *DevExpress*, al insertar campos de documento, los muestra como '{DOCVARIABLE <ValorDelCampo>}'.

[43] De igual forma, por defecto el Control de *DevExpress* muestra el valor del campo del documento en vez de su código; como el requisito es que se vea el código del campo y no el *tag* de las *OpenXML PowerTools* que tiene como valor, se tiene que hacer este remplazo explícitamente.

Modificaciones anteriores → Inserción del Campo tal cual se quiere que el usuario lo vea → Modificaciones posteriores.

A modo de ejemplo, se mostrará un escenario sobre cómo operaciones intermedias en la inserción de un Campo quedaban a la vista tras deshacer cambios. En la Figura 35, aparecería un Campo cualquiera insertado en el documento.

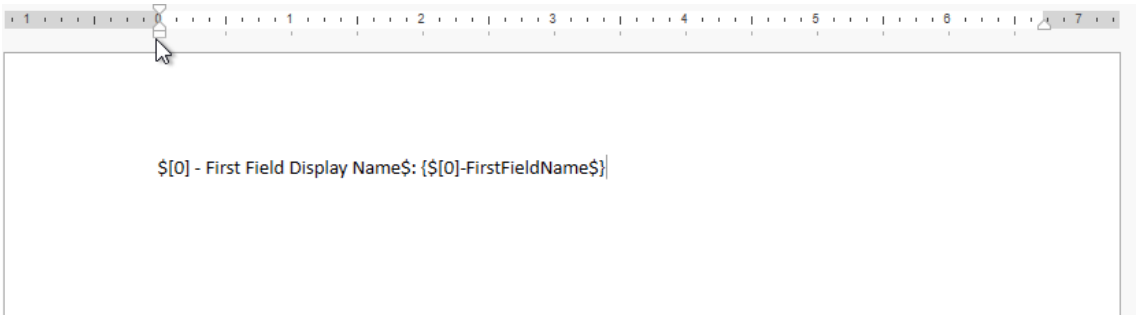

Figura 35. Ejemplo de Campo insertado en un documento.

Cuando el usuario del Diseñador, para dicho estado actual, invocaba el comando de deshacer, el Campo quedaba como en la Figura 36.

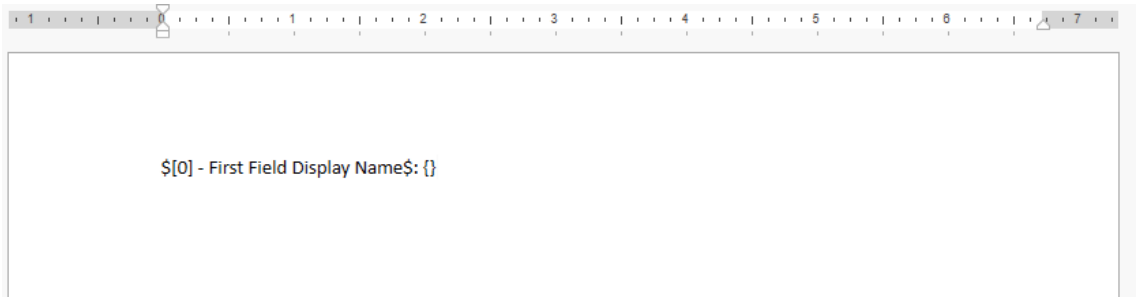

Figura 36. Estado del Campo insertado en la Figura 35 tras que el usuario invocase el comando de deshacer una vez.

Si tras esto, se volvía a invocar por segunda vez el comando de deshacer, el estado era el de la Figura 37, donde se muestra el *'DOCVARIABLE'* que *DevExpress* muestra por defecto.

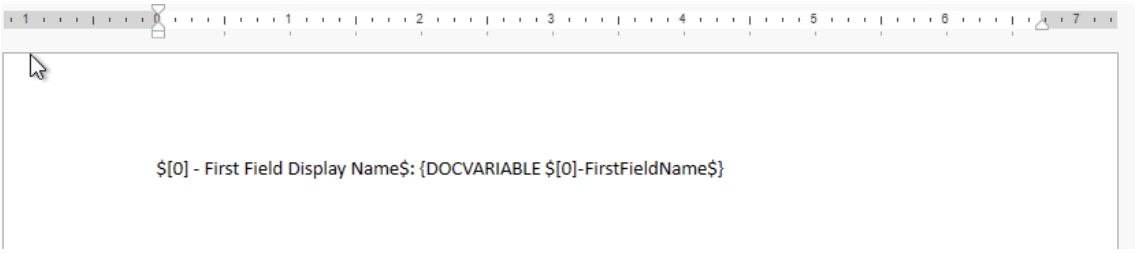

Figura 37. Estado del Campo insertado en la Figura 35 tras que el usuario invocase el comando de deshacer por segunda vez.

Tras una tercera invocación sobre el comando de deshacer los cambios, el valor del Campo, que se requería que se mantuviera oculto al usuario del Diseñador, queda expuesto en el mismo (Figura 38).

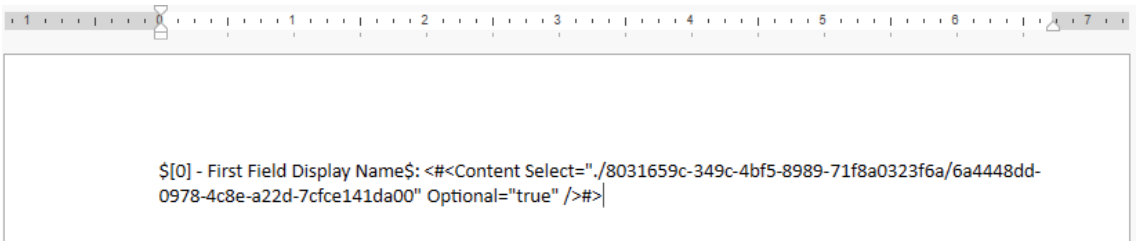

Figura 38. Estado del Campo insertado en la Figura 35 tras que el usuario invocase el comando de deshacer por tercera vez.

Si el usuario deshacía cambios de nuevo, el estado del Campo sería similar al de la Figura 36. Por último, tras una quinta invocación al susodicho comando, los cambios asociados a la inserción del Campo quedaban finalmente deshechos (Figura 39). En resumen, se requería que el estado del Campo pasase del mostrado en la Figura 35 al de la Figura 39 tras una única invocación sobre el comando de deshacer. De igual forma, el comando de rehacer haría lo propio sobre la inserción del Campo.

Figura 39. Estado del Campo insertado en la Figura 35 tras que el usuario invocase el comando de deshacer por quinta vez.

Además, estas operaciones intermedias quedaban también a la vista cuando, en vez de insertar el Campo, se copia y pega uno ya insertado. En este caso, los cambios mostrados en la Plantilla quedaban tal que:

Modificaciones anteriores → Insertar copia del Campo → Cambiar Valor pegado por Código → Modificaciones posteriores.

M2: Diseño

Dado que esta mejora parecía una modificación bastante común sobre el control de *DevExpress*, se empezó buscando en el foro de soporte de la empresa a ver si se proponían soluciones a problemáticas similares.

En el ticket T240247 [64], se habla de sobrescribir los comandos de "deshacer/rehacer" del control, de forma que se identifique cuando se ejecutan estos comandos y para qué elementos *HistoryItem* (entrada en el *stack* de cambios). En un principio, se pensó que podría ser una solución ventajosa ya que permitiría evaluar fácilmente si el *HistoryItem* era referido a un Campo del *DTO* de datos o modificaciones intermedias de los mismos; de esta forma, se podría hacer que éstas se ignorasen o se borraran. Sin embargo, el soporte de *DevExpress* desaconsejaba el uso y manejo del *History* (histórico/*stack* de cambios) de un documento para código de producción, ya que forma parte de la *API* interna y no se encuentra documentada; además, la solución propuesta no terminaba de ser suficiente para los requisitos que se tenían.

En el ticket T800928 [65], si bien trata sobre el caso del control para hojas de cálculo, aplica de igual forma ya que ambos trabajan sobre el mismo *DocumentHistory*: se recomienda envolver los cambios que se quiere que se traten como transacciones entre los métodos *Document.BeginUpdate()* y *Document.EndUpdate()*. Esta solución, que también se encontró en otros tickets [66] [67], no parecía ser efectiva tras las pruebas que se hicieron en el Diseñador.

Otra solución que se probó fue la de desactivar temporalmente el histórico de cambios del documento justo antes de insertar el Campo, para reactivarlo cuando terminasen las operaciones intermedias. Esto no era una solución suficiente, dado que, al no registrar la inserción del Campo como tal, no podría ser deshecha ni rehecha; por ello, se decidió seguir buscando otra solución que cumpliera con los requisitos más estrechamente.

Por último, se decidió hacer una implementación algo más elaborada, que se encargase de registrar cuándo se inicia el proceso de inserción de un Campo, registrar cuándo han terminado las modificaciones intermedias, y guardarse el rango correspondiente de cambios de forma que, cuando se quiera efectuar una operación de deshacer o rehacer sobre ese rango, aplicar el comando correspondiente recursivamente para dichas operaciones intermedias. Es decir, el conjunto de operaciones intermedias que se quieren ocultar al usuario quedaría registradas en una colección de rangos o pares de índices; cuando el usuario invoque el comando de deshacer o rehacer sobre el índice de uno de estos rangos, se desharán o reharán los *HistoryItems* comprendidos en ese rango.

Con esta implementación *'custom'*, se tuvo que abordar también el caso del copiado y pegado de texto del documento, evaluando si en el texto a pegar en el documento contiene algún Campo. Esto se hizo mediante una implementación *customizada* del comando de pegado o *'paste'*(la clase *CustomPasteSelectionCommand* de la Figura 41), de forma que se inspecciona el contenido del portapapeles en formato texto rico, sabiendo que, en este formato, los campos se identifican con la cadena *'field'*; en caso de que el portapapeles contuviera esta cadena, en primer lugar, se notifica el inicio de inserción de un Campo. Así, se continúa con la ejecución por defecto del comando, que básicamente inserta en el documento todo el contenido del portapapeles. Además, en este punto, para asegurar que los valores de los campos del documento (es decir, de los Campos del *DTO* insertados) quedan ocultos al usuario y que éste solo ve los códigos, se invoca explícitamente el método asociado del *IDocumentEditorControl* (recordamos, sería la clase encargada de implementar la lógica asociada al control del editor de documentos y que actualmente sería una implementación del *Rich Edit Control* de *DevExpress*). Por último, en la ejecución de esta versión personalizada del comando de pegado, en el caso de que el contenido del portapapeles contuviera un Campo, se notifica la finalización de inserción de Campos. El sentido y resultado de toda esta lógica, es que, si el contenido a pegar contiene un Campo, se trate todo ese contenido como un bloque a la hora de deshacer o rehacer cambios y que las operaciones intermedias pasen igualmente desapercibidas.

M2: Desafíos de programación

Dado que se tuvo que implementar un algoritmo concreto para sobrescribir el comportamiento de los comandos de deshacer y rehacer, como además para el de pegado en relación con las inserciones de Campos en el documento, el mayor reto sobre esta mejora fue el de ir construyendo y refinando este algoritmo.

En primer lugar, en busca ofrecer apoyo a la hora de comprender la problemática y la solución implementada, conviene explicar brevemente el significado del "índice actual de cambios" del *DocumentHistory*. Para el caso, podríamos imaginar el stack de cambios del histórico como un *array* que empieza en '-1' y con el siguiente contenido (Figura 40):

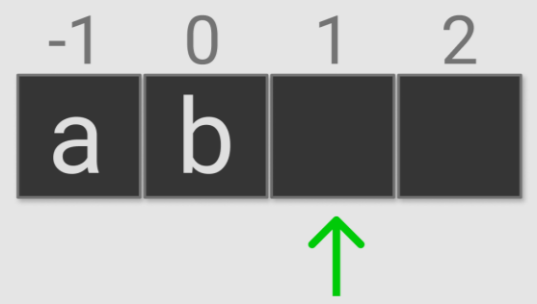

Figura 40. Ejemplo de una posible representación del stack de cambios del DocumentHistory, donde aparecen dos cambios, 'a' y 'b', el índice actual de cambios representado por una flecha verde, y los correspondientes índices en la parte superior.

Al empezar una "sesión" (abrir un nuevo documento, crear uno nuevo, etc.), el índice actual de cambios sería '-1', indicando que actualmente no existe ningún cambio para la sesión actual. En la representación de la Figura 40, se habrían insertado los cambios 'a' y 'b', y por ello, el índice actual de cambios, representado por la flecha verde, tendría el valor de '1'. Si con este estado actual el usuario invoca el comando de deshacer, el índice actual de cambios pasaría a tener el valor '0' y se desharía el cambio 'b'.

Teniendo esto en cuenta, podríamos decir que la solución se fundamenta en la clase *DocumentHistoryWrapper*, que como su nombre indica, sería una clase envoltorio de *DocumentHistory*, para el caso del Control de *DevExpress*. En resumidas palabras, esta clase ofrece una serie de facilidades para trabajar con el histórico de cambios del Control, principalmente exponiendo los métodos para registrar los inicios y finalizaciones de inserciones de Campos, para poder registrar los rangos de índices, y encapsular la lógica para realizar la consulta de si el índice actual del histórico estaría relacionado con una inserción de un Campo (en otras palabras, si es una operación intermedia). En la Figura 41 se puede observar el diagrama de clases de esta solución.

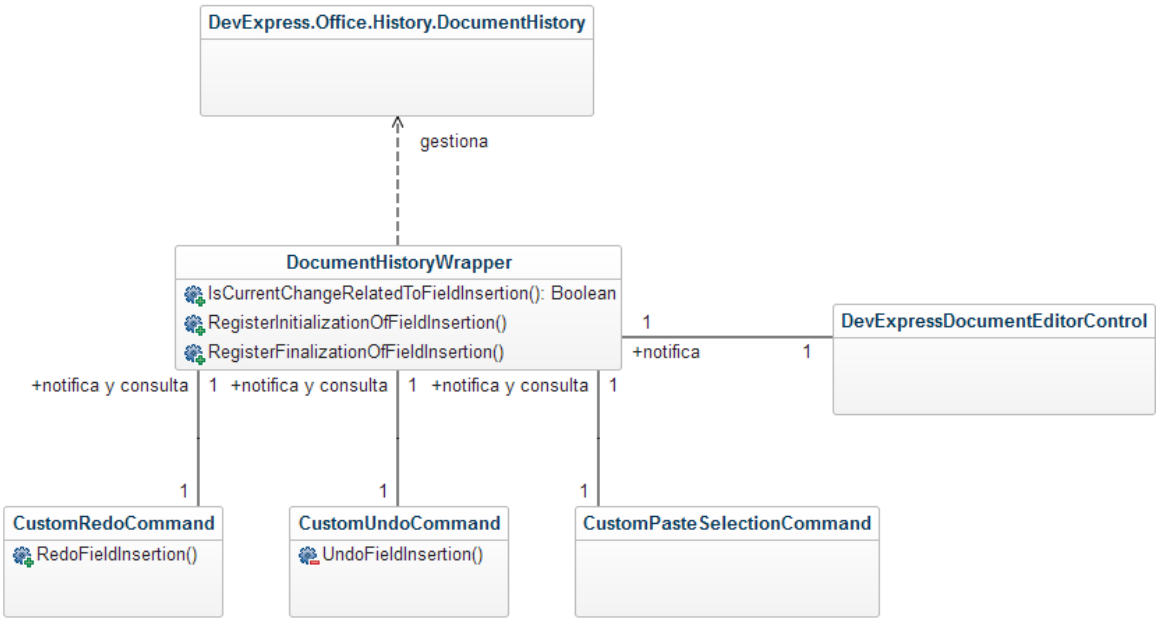

Figura 41. Diagrama de clases sobre la mejora "M2: Ocultar las operaciones intermedias con los comandos de deshacer y rehacer para inserciones de Campos de la Plantilla".

En resumidas cuentas, la clase que implementa *IDocumentEditorControl*, *DevExpressDocumentEditorControl*, sería la encargada de notificar al *DocumentHistoryWrapper* sobre los inicios y finalizaciones de inserciones de Campos; de esta forma, la clase envoltorio del

histórico del documento podría llevar un seguimiento de cuáles serían los rangos de índices de las operaciones intermedias. Dependiendo de si el usuario invoca el comando de deshacer, rehacer o pegado sobre el inicio o final de un rango de operaciones intermedias, se lleva a cabo un comportamiento u otro:

- Deshacer o rehacer: Se evalúa si, para la operación requerida, el índice de cambio actual corresponde con una operación intermedia para deshacer todo el rango recursivamente.
- Pegar: Se evalúa si el contenido del portapapeles contiene Campos, de forma que se registra todo el contenido como un único cambio; así, se deshará o rehará en bloque.

M2: Pruebas

De igual forma que ocurría con las anteriores mejoras, todavía no se habría propuesto la iniciativa de mejorar la cobertura de las pruebas automatizadas en los proyectos del equipo de trabajo; por ello, las pruebas que se llevaron a cabo sobre la presente mejora fueron única y exclusivamente manuales.

A lo largo del desarrollo, la casuística más repetida con la que se iba probando la lógica implementada, era la de, abriendo el Diseñador de Plantillas mediante los *ManualTests*, insertar un Campo cualquiera o una sucesión de ellos, e ir probando a deshacer y rehacer cambios. Esto, apoyado con las facilidades de *Visual Studio* para acoplar el *debugger* a un proceso concreto, fueron las principales pruebas que se hicieron. Algunas variaciones sobre esto, fue la de ir añadiendo cambios que no fueran operaciones intermedias, como puede ser el caso de texto plano; el objetivo era el de probar más casuísticas con respecto del reconocimiento y tratamiento que se hacía de los índices asociados a inserciones de Campos.

Para el caso del comando de pegar, la casuística empleada solía ser mezclar párrafos con texto plano y algún Campo insertado, probando a pegar reiteradas veces ese párrafo o varios de ellos, e ir ejecutando los comandos de deshacer y rehacer comprobando que en todo momento las operaciones intermedias quedaban ocultas al usuario del Diseñador.

M3: Soporte a localización y traducción de Plantillas

Esta mejora trata de ofrecer la capacidad de que los diferentes Elementos que aparecen modelados en las Plantillas asociadas a *Reports* puedan ser localizados a una Cultura especificada (por ejemplo, español de Argentina o inglés británico). Además, esta localización de Elementos se fundamente en el hecho de que las Plantillas se persisten con los identificadores de los Campos insertados, no con sus *Display Names*, Descripciones o Nombres. Así, estas propiedades de los Campos quedan siempre actualizadas, ya que al extraer de persistencia la Plantilla para generar el *Report*, los identificadores de los Campos son siempre los mismos, independientemente de que alguna de las propiedades de los mismos haya sido cambiada desde el momento en el que se añadió a la Plantilla en sí.

M3: Especificación de requisitos

Principalmente, lo que se pedía de esta mejora era que dada la actual funcionalidad que exponía *Reports Designer* de recuperar una Plantilla resuelta, que ésta se devolviera también ya localizada. De esta forma, ante la petición sobre el motor de *Reports* de generar un *Report* propiamente dicho, la orquestación solo tiene que preocuparse de obtener la Plantilla para dicha petición del propio *Reports Designer* para entonces pasársela al motor; esto es de esta forma, porque el motor no entiende de identificadores o Campos, simplemente busca en el documento (la Plantilla) *tags* de las *OpenXML PowerTools* para remplazar los datos del *XML* de datos.

Con esta premisa, las casuísticas que encontramos en una Plantilla para traducir dependen del tipo de los Elementos que se han insertado:

- Para los Elementos que vayan entre delimitadores de identificadores, '$', simplemente sería remplazarlos por su cadena de caracteres localizada obtenida, eliminando además los delimitadores de identificadores. Estos delimitadores se usan para que, tanto el Diseñador de Plantillas como *Reports Designer*, puedan saber dónde se encuentran los Elementos que requieren de tratamiento (en este caso, que requieren de localización); sin embargo, no son relevantes para el usuario ni el motor. Ejemplos de estos Elementos serían *Display Names*.
- En el caso de que estos Elementos entre 'delimitadores de identificadores' formen parte de un Campo de tipo Enumerado, en vez de por su *Display Name* se tendrán que remplazar por la Descripción del mismo. Esto es así, porque en los Campos de tipo Enumerado, es la Descripción la que contiene la información relevante para el enunciado de éste.
- Para los Elementos cuyos identificadores se encuentren en *tags*, habría que localizarlos como el Nombre del Campo ya que serían parte del camino en *XPath* que selecciona un Campo del *DTO* de entrada de datos.

M3: Diseño

Para el diseño y arquitectura de esta mejora, hay que tener en cuenta algunos detalles técnicos que son de importancia para el contexto de la misma. La principal problemática, es que para obtener el Campo asociado a un identificador y por ende a alguna de sus propiedades (*Display Name*, Descripción o Nombre), es necesario obtener el ensamblado al que pertenece dicho Campo. En este punto, para localizar una Plantilla hay dos opciones:

- Referenciar todos los ensamblados que contengan *DTO* que puedan ser usados para modelar *Reports.*
- De manera dinámica, cargar únicamente el ensamblado concreto al que pertenecen los Elementos de la Plantilla actual.

Por cuestiones de eficiencia y diseño, se eligió la segunda opción. Para saber a qué ensamblado corresponden los Elementos de la Plantilla se pudo utilizar la información de persistencia, ya que una Plantilla se persiste con la referencia al Módulo al que pertenecen sus Elementos (o, en otras palabras, dónde ésta fue modelada). Dado que un Módulo contiene el nombre de su ensamblado de Contratos que contiene los *DTOs*, simplemente bastaría con acceder a esa referencia. Así, con el nombre del ensamblado de Contratos, solo falta la manera de obtener dicho ensamblado y cargarlo dinámicamente para acceder a sus Elementos. Para ello, se hace uso del microservicio de *Deployment*, el cual tiene información sobre las diferentes versiones de los microservicios del producto; una vez obtenida la última versión del Módulo con una petición a *Deployment*, se obtiene el ensamblado de Contratos correspondiente a dicho Módulo y se instancia mediante un contexto de carga de ensamblados *custom (AssemblyLoadContext* [68]*)*. De esta forma, y como puede observarse en la Figura 42, el *TemplateLocalizer* hace uso del *IElementLocalizationProvider;* la implementación de dicha interfaz, *GlobalResourcesLocalizationProvider*, sería la encargada de proveer de la localización de un Elemento haciendo uso del *Global Resources* correspondiente al ensamblado de Contratos mencionado anteriormente.

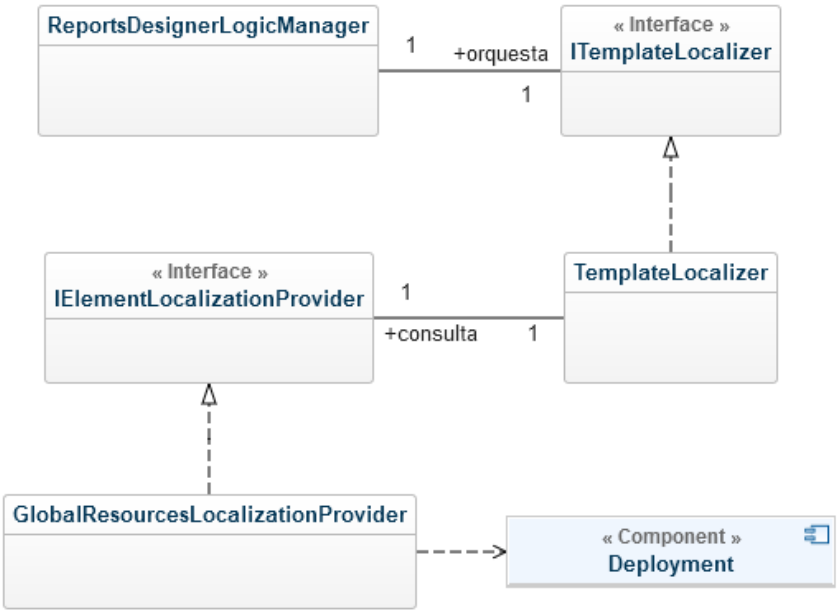

Figura 42. Diagrama que muestra los aspectos más relevantes del diseño de la mejora M3.

El documento de la Plantilla se va analizando, párrafo a párrafo, buscando los tipos de Elementos que se comentaban anteriormente de forma que, por cada uno de ellos, se vayan aplicando los remplazos y modificaciones necesarias sobre la Plantilla con las cadenas localizadas que se han ido obteniendo.

Así, toda esta lógica se configura en un módulo denominado *TemplateLocalizer*, el cual es usado por el *manager* de la lógica de *Reports Designer* en la acción pública mediante una interfaz *ITemplateLocalizer*; de esta forma, el *manager* de la lógica no conoce los detalles de implementación de dicho módulo, solo el contrato que expone para su uso y la información de entrada que este necesita para funcionar. Además, dado que la Cultura a la cual localizar es un parámetro de entrada de la acción pública que se expone, se comprueba que dicha Cultura es una cultura válida especificada según el estándar RFC 4646 [8] [69]. De esta forma, la Cultura requerida para la localización de los Elementos de la Plantilla se acaba pasando como argumento al método que obtiene la propiedad requerida para el identificador de un Elemento.

M3: Desafíos de programación

En esta mejora, el desafío de programación más importante ha sido la iniciativa de comenzar a practicar *TDD*. Dejando de lado la dificultad que implica empezar con dicha técnica en un proyecto real y como primer acercamiento a la misma, se tuvo que lidiar con que el proyecto no contaba con los tests suficientes como para trabajar de forma más cómoda. Esto hizo que aparecieran algunos fallos durante el tramo final del desarrollo de la mejora, en los que se tuvo que invertir cierto tiempo en encontrar su defecto y solución.

Otra complicación que hubo fue la de configurar la conexión con *Deployment* para la obtención del Módulo y ensamblado, dado que fue una de las primeras veces que se tuvo que abordar una cuestión parecida al respecto de infraestructura.

M3: Pruebas

Como ya se ha ido adelantando a lo largo del trabajo al respecto de las pruebas o tests sobre la propuesta del módulo Diseñador de Plantillas, uno de los mayores esfuerzos y logros ha sido la iniciativa de empezar a aplicar *TDD* sin experiencia previa, sobre la implementación de una funcionalidad real y con resultados satisfactorios.

El acercamiento en particular sobre *TDD* que se ha seguido, ha sido el de verificar el comportamiento del sistema mediante la especificación de escenarios o casuísticas. Así, los test

sirven como especificación ejecutable del comportamiento que exhibe el sistema: mientras que no cambie el comportamiento del sistema, la implementación se puede cambiar o refactorizar tanto como se quiera, que siempre y cuando no cambie dicho comportamiento, los test seguirán pasando. En otras palabras, los test no están ligados a la implementación, sino al comportamiento que exhibe el sistema.

Los test implementados han sido mayoritariamente unitarios, aunque también ha habido algunos de integración. Por ello, y como la acepción de "de integración" puede depender del equipo en el que se esté trabajando, conviene mencionar qué significado tiene en el contexto de este trabajo. En nuestro caso, entendemos como tests de integración aquellos que prueban la interacción entre varias unidades funcionales; dichas unidades funcionales serían tanto sistemas externos al actual (bases de datos, redes, sistema de archivos, etc.) como otros módulos dentro del mismo sistema (ejemplos para la mejora actual serían el módulo de resolución de Plantillas y el de localización).

Test sobre remplazo del identificador de un Elemento por su Display Name

Uno de los primeros tests que se hizo, fue el de probar que, dada una Plantilla con un Elemento entre *tags* delimitadores de identificadores '$', al pasarla por el módulo de localización, la Plantilla resultante tendría el identificador de dicho Elemento remplazado por su correspondiente *Display Name*. Para más detalle, podemos observar la Figura 43 donde aparece la documentación y nombre asociado al test.

```
/// <summary>
///     The method:
///     <see cref="ITemplateLocalizer.LocalizeAsync(WordprocessingDocument,Guid,string,CultureInfo)"/>,
///     when a Template with both text and a single Display Name identifier -that is
///     recognized- is received, then the output Template has the Display Name identifier
///     replaced by its value.
/// </summary>
/// <returns> An empty task that enables this method to be awaited. </returns>
/// <remarks>
///     For an identifier to be "recognized" means that it can be provided by an <see cref="IElementLocalizationProvider"/>.
/// </remarks>
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 17 days ago | 1 author, 2 changes
public async Task Localize_TemplateWithDisplayNameIdentifier_TemplateWithReplacedDisplayNameValue()
```

Figura 43. Documentación y cabecera del test sobre el remplazo de identificadores de Elementos por su Display Name.

Para evitar entrar en detalle de cómo se provee la localización del Elemento, y dado que el test solo está interesado en probar el reconocimiento y remplazo de un identificador de un Elemento entre '$' por su *Display Name*, se acabó falseando la implementación del *'IElementLocalizationProvider'* mediante el uso de un *stub*[44]; de esta forma, se configuró este doble de test para que, cuando se le preguntase por la localización del identificador del Elemento de la Plantilla en cuestión, devolviera directamente su valor localizado. El detalle de la configuración de dicho *stub* así como del resto del test, puede observarse en la Figura 44.

[44] Un *stub* es un doble de test que se encarga de suplantar una implementación real proporcionando respuestas predefinidas e invariables ante invocaciones a sus miembros [86].

```
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 17 days ago | 1 author, 2 changes
public async Task Localize_TemplateWithDisplayNameIdentifier_TemplateWithReplacedDisplayNameValue()
{
    // Arrange.
    Mock<IElementLocalizationProvider> elementLocalizationProviderStub = new Mock<IElementLocalizationProvider>();
    Guid templateDisplayNameIdentifier = Guid.Parse("5cf6497e-15a4-4926-9315-10717024e1dd");

    elementLocalizationProviderStub
        .Setup(mock => mock.GetLocalizedElementAsync(
            templateDisplayNameIdentifier,
            It.IsAny<Guid>(),
            DocumentationProperty.DisplayName,
            It.IsAny<string>(),
            null))
        .ReturnsAsync(() => "Display Name Localized String Value");

    ITemplateLocalizer templateLocalizer = new TemplateLocalizer(elementLocalizationProviderStub.Object);

    WordprocessingDocument templateWithDisplayNameIdentifier =
        GetTemplateFromPath("./TestFiles/TemplateLocalizer/TemplateWithDisplayNameIdentifier.docx");

    // Act.
    WordprocessingDocument localizedTemplateWithDisplayNameValue = await templateLocalizer.LocalizeAsync(
        templateWithDisplayNameIdentifier,
        FakeTemplateModuleId,
        FakeTemplateSourceAssemblyName).ConfigureAwait(false);
```

Figura 44. 'Arrange' y 'Act' del test sobre el remplazo de identificadores de Elementos por su Display Name.

Por último, decir que la verificación que realiza el test sobre la localización de Plantillas es la de comparar la Plantilla modificada por el módulo de localización, con una Plantilla esperada, en la que el Elemento de la Plantilla original aparece con su *Display Name;* esto puede observarse en la Figura 45, mediante el uso de una clase de utilidad para comparar documentos mediante las *OpenXML PowerTools*.

```
// Assert.
byte[] expectedTemplateWithDisplayNameValuesByteArray =
    File.ReadAllBytes("./TestFiles/TemplateLocalizer/ExpectedTemplateWithDisplayNameValue.docx");

WmlDocument expectedTemplateWithDisplayNameValue =
    new WmlDocument("Expected Template With Display Name Value", expectedTemplateWithDisplayNameValuesByteArray);

OpenXmlPowerToolsDocumentComparer.CompareAssembledDocumentWithExpectedAndCheckValidationErrors(
    resolvedDocument: localizedTemplateWithDisplayNameValue,
    resolvedDocumentName: "Localized Template With Display Name Value",
    expectedDocument: expectedTemplateWithDisplayNameValue);
```

Figura 45. 'Assert' del test sobre el remplazo de identificadores de Elementos por su Display Name.

Test sobre remplazo del identificador de un Campo por su Nombre

En este caso, se nos plantea el escenario de una Plantilla que contiene el identificador de un Campo dentro de su correspondiente tag en el formato de las *OpenXML PowerTools*, y que, tras ser localizada, la Plantilla debería contener el Nombre del Campo. La documentación en detalle del test, así como su nombre, podemos encontrarlo en la Figura 46.

```
/// <summary>
///     The method:
///     <see cref="ITemplateLocalizer.LocalizeAsync(WordprocessingDocument,Guid,string,CultureInfo)"/>,
///     when a Template with both text and a single Field Name identifier -that is
///     recognized- is received, then the output Template has the Field Name identifier
///     replaced by its value.
/// </summary>
/// <returns> An empty task that enables this method to be awaited. </returns>
/// <remarks>
///     For an identifier to be "recognized" means that it can be provided by an <see cref="IElementLocalizationProvider"/>.
/// </remarks>
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 17 days ago | 1 author, 2 changes
public async Task Localize_TemplateWithFieldNameIdentifier_TemplateWithReplacedFieldValue()
```

Figura 46. Documentación y cabecera del test sobre el remplazo de identificadores de Campos por su Nombre.

De igual forma al caso anterior, no estamos interesados en verificar el mecanismo de obtención de la localización del Elemento, sino más bien de su correcta detección en la Plantilla y remplazo. Por ello, se vuelve a hacer uso de un *stub* para configurar el hecho de que, ante una petición al *IElementLocalizationProvider* de proveernos de la localización de un Elemento dado su identificador, si ese identificador es exactamente el que nosotros esperamos y que corresponde al Campo a localizar, que directamente nos devuelva lo que esperamos como su Nombre. Esto puede observarse en la Figura 47.

```
public async Task Localize_TemplateWithFieldNameIdentifier_TemplateWithReplacedFieldValue()
{
    // Arrange.
    Mock<IElementLocalizationProvider> elementLocalizationProviderStub = new Mock<IElementLocalizationProvider>();
    Guid templateFieldNameIdentifier = Guid.Parse("211030b4-9388-4d0b-98aa-5b9ddc9f366b");

    elementLocalizationProviderStub
        .Setup(stub => stub.GetLocalizedElementAsync(
            templateFieldNameIdentifier,
            It.IsAny<Guid>(),
            DocumentationProperty.Name,
            It.IsAny<string>(),
            null))
        .ReturnsAsync(() => "FieldNameLocalizedStringValue");

    ITemplateLocalizer templateLocalizer = new TemplateLocalizer(elementLocalizationProviderStub.Object);

    WordprocessingDocument templateWithFieldNameIdentifier =
        GetTemplateFromPath("./TestFiles/TemplateLocalizer/TemplateWithFieldNameIdentifier.docx");

    // Act.
    WordprocessingDocument localizedTemplateWithFieldNameValue = await templateLocalizer.LocalizeAsync(
        templateWithFieldNameIdentifier,
        FakeTemplateModuleId,
        FakeTemplateSourceAssemblyName).ConfigureAwait(false);
```

Figura 47. 'Arrange' y 'Act' del test sobre el remplazo de identificadores de Campos por su Nombre.

Por último, la verificación de este test es casi idéntica a la del test anterior, con la diferencia de que la Plantilla esperada contiene el mismo Campo que la Plantilla inicial, pero con el Nombre del mismo en lugar del identificador. Esta verificación puede observarse en la Figura 48.

```
// Assert.
byte[] expectedTemplateWithFieldNameValueByteArray =
    File.ReadAllBytes("./TestFiles/TemplateLocalizer/ExpectedTemplateWithFieldValue.docx");

WmlDocument expectedTemplateWithFieldNameValue =
    new WmlDocument("Expected Template With Field Name Value", expectedTemplateWithFieldNameValueByteArray);

OpenXmlPowerToolsDocumentComparer.CompareAssembledDocumentWithExpectedAndCheckValidationErrors(
    resolvedDocument: localizedTemplateWithFieldNameValue,
    resolvedDocumentName: "Localized Template With Field Name Value",
    expectedDocument: expectedTemplateWithFieldNameValue);
```

Figura 48. 'Assert' del test sobre el remplazo de identificadores de Campos por su Nombre.

Test sobre la localización de un Elemento el cual no es reconocido

Tal y como se ha podido apreciar en la Figura 43 y Figura 46, el *remarks* de la documentación, menciona que el significado de que un identificador (asociado a un Elemento) no sea reconocido, es el hecho de que no pueda ser provisto por el *IElementLocalizationProvider*. Bajo esta casuística, el comportamiento deseado para el módulo de localización es que lance una excepción *FunctionalValidationException* tal y como indica el nombre del test en la Figura 49.

```
/// <summary>
///     The method:
///     <see cref="ITemplateLocalizer.LocalizeAsync(WordprocessingDocument,Guid,string,CultureInfo)"/>,
///     when a Template with a single Element identifier -that is not recognized- is
///     received, then a <see cref="FunctionalValidationException"/> is thrown stating that
///     the Element couldn't be retrieved.
/// </summary>
/// <remarks>
///     For an identifier to be "recognized" means that it can be provided by an <see cref="IElementLocalizationProvider"/>.
/// </remarks>
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 13 days ago | 1 author, 3 changes
public void Localize_TemplateWithNotRecognizedElementIdentifier_ThrowsFunctionalValidationException()
```

Figura 49. Documentación y cabecera del test sobre la localización de un Elemento el cual no es reconocido.

Más concretamente, se espera que la excepción lanzada por el módulo de localización contenga un mensaje en específico, que sea descriptivo de porqué ha ocurrido dicha excepción. Este mensaje puede observarse en la Figura 50, además de cómo se configura el doble de test del *IElementLocalizationProvider* para que, en el caso de que se le pida proveer la localización de cualquier Elemento, siempre genere una *InvalidOperationException*; este sería el comportamiento que exhibiría el *IElementLocalizationProvider* en caso de que no reconociese un Elemento. Otro elemento diferenciador de este test respecto a los mostrados anteriormente es que no ejecuta directamente la llamada al módulo de localización, si no que crea un delegado para que sea invocado más adelante; esto es para que, en el apartado de verificaciones del test, se pueda comprobar si lanza la excepción esperada con su mensaje correspondiente.

```
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 13 days ago | 1 author, 3 changes
public void Localize_TemplateWithNotRecognizedElementIdentifier_ThrowsFunctionalValidationException()
{
    // Arrange.
    Mock<IElementLocalizationProvider> elementLocalizationProviderMock = new Mock<IElementLocalizationProvider>();

    // If a requested Element is not found, an InvalidOperationException will be thrown.
    elementLocalizationProviderMock.Setup(mock => mock.GetLocalizedElementAsync(
        It.IsAny<Guid>(),
        It.IsAny<Guid>(),
        It.IsAny<DocumentationProperty>(),
        It.IsAny<string>(),
        It.IsAny<CultureInfo>()))
    .ThrowsAsync(new InvalidOperationException());

    ITemplateLocalizer templateLocalizer = new TemplateLocalizer(elementLocalizationProviderMock.Object);

    WordprocessingDocument templateWithElementIdentifier =
        GetTemplateFromPath("./TestFiles/TemplateLocalizer/TemplateWithDisplayNameIdentifier.docx");

    // Act.
    Func<Task> localizeTemplateDelegate = async () => await templateLocalizer.LocalizeAsync(
        templateWithElementIdentifier,
        FakeTemplateModuleId,
        FakeTemplateSourceAssemblyName).ConfigureAwait(false);

    // Assert.
    const string ExpectedMessage =
        @"""The Element with the identifier '5cf6497e-15a4-4926-9315-10717024e1dd' couldn't be localized.""";

    localizeTemplateDelegate.Should().Throw<FunctionalValidationException>().WithMessage(ExpectedMessage);
}
```

Figura 50. Implementación completa del test sobre la localización de un Elemento el cual no es reconocido.

Test sobre localización del identificador de un Elemento en su Display Name

Para este test, mostraremos el caso de un test de integración que integra el uso del *IElementLocalizationProvider* con el módulo de localización, donde se plantea el escenario de localizar un Elemento en su correspondiente *Display Name*, tal y como se especifica en la Figura 51.

```
/// <summary>
///     The method:
///     <see cref="TemplateLocalizer.LocalizeAsync(WordprocessingDocument,Guid,string,CultureInfo)"/>,
///     when a Template with both text and a single Display Name with its associated identifier
///     -that is recognized- is received, then the output Template has the Display Name identifier
///     replaced by its localized string value.
/// </summary>
/// <returns> An empty task that enables this method to be awaited. </returns>
/// <remarks>
///     For an identifier to be "recognized" means that it can be provided by an <see cref="IElementLocalizationProvider"/>.
/// </remarks>
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 13 days ago | 1 author, 1 change
public async Task Localize_TemplateWithDisplayNameIdentifier_TemplateWithLocalizedDisplayNameValue()
```

Figura 51. Documentación y cabecera del test sobre localización de un Element en su correspondiente Display Name.

Tal y como puede observarse en la Figura 52, en este caso no hay ninguna configuración de algún doble de tests para falsear el *IElementLocalizationProvider;* sin embargo, conviene mencionar que, para evitar hacer la conexión con *Deployment* tal y como se explicaba en el apartado de "M3: Diseño", sí que se ha registrado un *fake*[45] mediante inyección de dependencias para simular el comportamiento que tendría *Deployment* al respecto de las peticiones que se le hagan.

```
[TestMethod]
| 0 references | Alejandro Lozano Jiménez, 13 days ago | 1 author, 1 change
public async Task Localize_TemplateWithDisplayNameIdentifier_TemplateWithLocalizedDisplayNameValue()
{
    // Arrange.
    TemplateLocalizer templateLocalizer = new TemplateLocalizer(GetElementLocalizationProviderInstance());

    WordprocessingDocument templateWithDisplayNameIdentifier =
        GetTemplateFromPath("./TestFiles/TemplateLocalizer/TemplateWithDisplayNameIdentifier.docx");
    Guid dummyTemplateModuleId = Guid.Empty;
    string dummyTemplateSourceAssemblyName = "Dummy Template Source Assembly Name";

    // Act.
    WordprocessingDocument localizedTemplateWithDisplayNameValue = await templateLocalizer.LocalizeAsync(
        templateWithDisplayNameIdentifier,
        dummyTemplateModuleId,
        dummyTemplateSourceAssemblyName).ConfigureAwait(false);

    // Assert.
    byte[] expectedTemplateWithDisplayNameValueByteArray =
        File.ReadAllBytes("./TestFiles/TemplateLocalizer/ExpectedTemplateWithDisplayNameValue.docx");

    WmlDocument expectedTemplateWithDisplayNameValue =
        new WmlDocument("Expected Template With Display Name Identifier", expectedTemplateWithDisplayNameValueByteArray);

    OpenXmlPowerToolsDocumentComparer.CompareAssembledDocumentWithExpectedAndCheckValidationErrors(
        resolvedDocument: localizedTemplateWithDisplayNameValue,
        resolvedDocumentName: "Localized Template With Single Display Names Value",
        expectedDocument: expectedTemplateWithDisplayNameValue);
}
```

Figura 52. Implementación completa del test sobre localización de un Element en su correspondiente Display Name.

Listado de test realizados sobre el módulo de localización de Plantillas

A continuación, pasaremos a enumerar todos los tests que se han llevado a cabo sobre el módulo de localización de Plantillas. En primer lugar, se listarán los test unitarios en la Figura 53.

[45] Un *fake* es un doble de test que falsea o imita el comportamiento de su contraparte real, pero simplificando detalles o sustituyendo implementaciones que son más eficientes temporalmente hablando [86] [87].

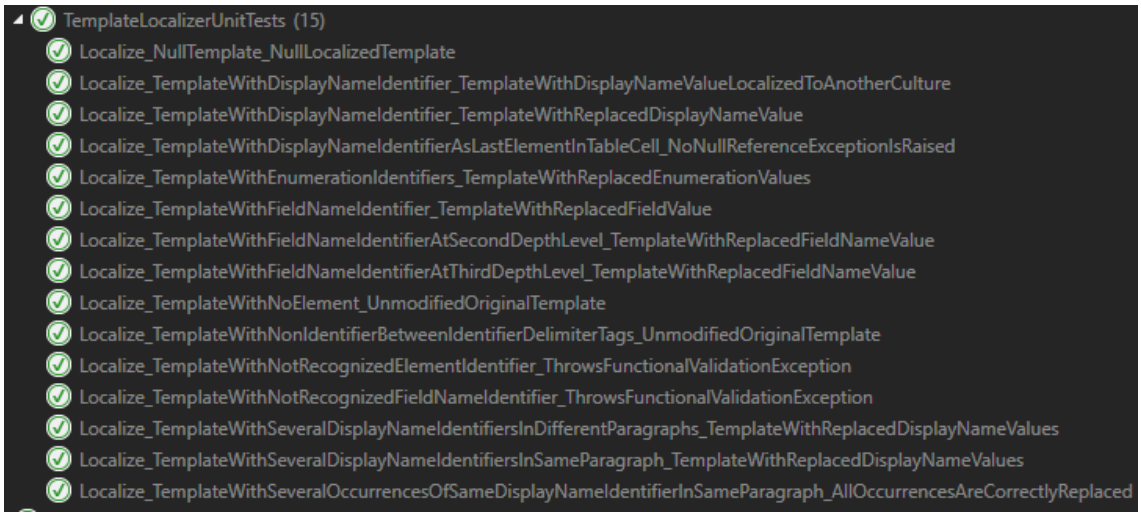

Figura 53. Listado de test unitarios sobre el módulo de localización de Plantillas.

Los test de integración del módulo de localización son los que se muestran en la Figura 54.

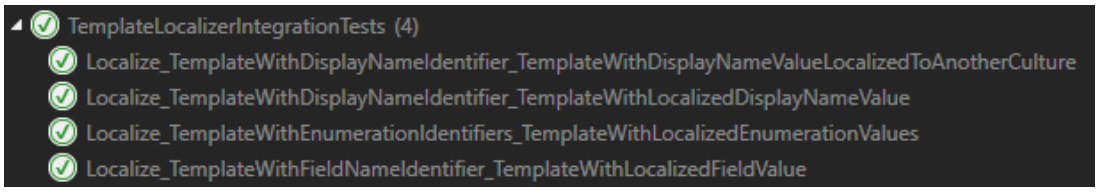

Figura 54. Listado de tests de integración sobre el módulo de localización de Plantillas.

Además, también se realizaron test unitarios sobre el manager de la lógica de *Reports Designer*, con la intención de verificar que se hace un correcto tratamiento de los argumentos de entrada con relación de la Cultura. Estos tests serían los que aparecen en la Figura 55.

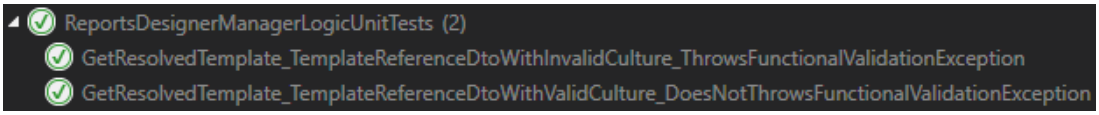

Figura 55. Listado de tests unitarios sobre el mánager de la lógica de Reports Designer.

Por último, listar los test de integración que se hicieron sobre dicho manager para probar la configuración de la llamada sobre el módulo de localización. Estos test se muestran en la Figura 56.

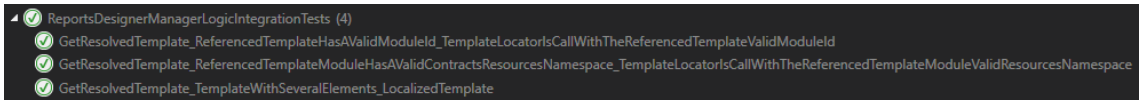

Figura 56. Listado de tests de integración sobre el mánager de la lógica.

M4: Mejoras visuales

Como se adelantaba en los objetivos del presente trabajo, también se han realizado algunas mejoras visuales sobre el Diseñador de Plantillas. Como realmente su alcance es muy reducido, se comentarán sus detalles brevemente.

M4: Grid de DTO de datos redimensionable

Dado que los analistas encargados de modelar *Reports* estaban encontrado problemas en el caso de *DTOs* de datos con Campos con nombres muy largos, se decidió dar soporte a que el apartado de la vista en el que se encuentra dicho *TreeView* fuera redimensionable. De esta forma,

no sería necesaria la tediosa tarea de hacer uso de la barra de *scroll* lateral, tal y como aparece en la Figura 57. Como se puede observar en la Figura 58, se permitió el redimensionado del *Grid* para que los nombres de Campos con gran longitud pudieran leerse sin mayor dificultad.

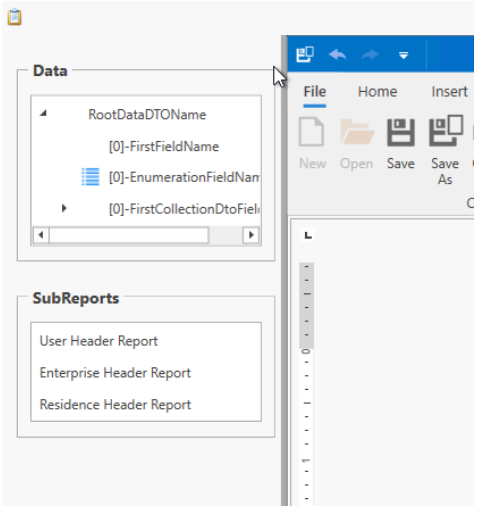

Figura 57. Vista del TreeView donde aparece el DTO de datos de entrada del Report, y donde aparece la barra de scroll lateral dada la longitud del nombre de algunos campos.

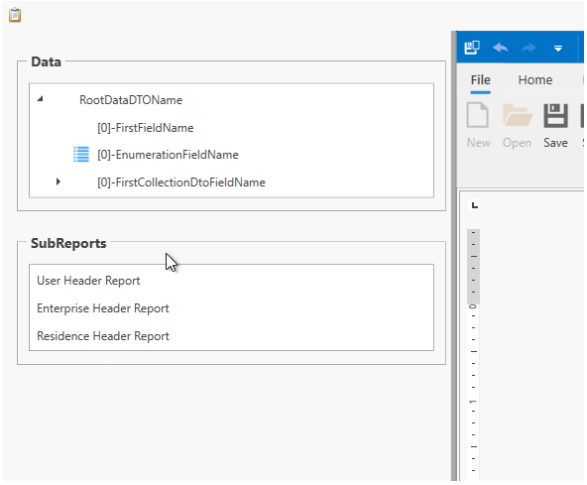

Figura 58. Vista del TreeView donde aparece el DTO de datos de entrada del Report, y donde no aparece la barra de scroll lateral.

La implementación consistió en simplemente modificar una propiedad del *Grid* en el que se encontraba tanto el *TreeView* que aloja el *DTO* de datos, y el Control de *DevExpress* para la edición de documentos (Plantillas) de texto rico.

M4: Iconos que ayudan a diferenciar Campos de tipo Enumerado

Otra modificación para mejorar el diseño de Plantillas en el Diseñador fue el de indicar el tipo de los Campos de tipo Enumerado mediante un icono en el *TreeView*. Este icono y consecuente mejora puede apreciarse en la Figura 57 y la Figura 58, como para uno de los Campos del *DTO* aparece un pequeño icono representando su tipo. Recordar que, si bien en dichas figuras puede parecer obvio el tipo del Campo porque se especifica en su nombre, el *DTO* de datos pertenece al proyecto de *ManualTest*, y se usa un *DTO* falso con nombres descriptivos de qué representan sus Campos; en el caso de Campos de un *DTO* real, puede que en muchos casos no quede claro que se trata de un Enumerado. Además, visualmente se puede identificar mejor el tipo sin tener que leer el propio nombre.

Como detalles de implementación, no existe ninguno relevante. Únicamente se tuvo que realizar unas pequeñas modificaciones en el *XAML* de la vista para que, en el caso de que uno de los elementos del *TreeView* fuera de tipo Enumerado, se mostrara un icono leído de los recursos del proyecto.

4.2.4 Desafíos de programación comunes a todas las mejoras

Tal y como se ha mencionado anteriormente, el contexto de este proyecto es el del desarrollo de un *ERP* de nueva generación, basado en técnicas de modelado y generación de código, y arquitectura basada en microservicios. Por lo tanto, uno de los mayores desafíos que se ha tenido que afrontar durante el desarrollo de las diferentes mejoras, dada la poca experiencia en este contexto, era el de ir asimilando los diferentes conceptos y tecnologías asociadas mientras se iba desarrollando la nueva funcionalidad. Sin embargo, en la mayoría de los casos ocurría que las mejoras solían ser focalizadas, es decir, sobre un aspecto concreto, lo que permitía explorar y aprender sobre un único aspecto al mismo tiempo.

Respecto de la inyección de dependencias [70], tampoco se tenía especial conocimiento, por lo que al principio también supuso un obstáculo en la implementación de las funcionalidades. Tras algo de práctica, se acabó haciendo uso de la misma sin mayor problema, tanto para registrar nuevas dependencias en los proyectos de producción como en los de tests para registro de dobles de tests [71].

Ejemplo de esto último y de lidiar con código autogenerado tal y como se comentaba anteriormente, es la ampliación del registro de dependencias que se hizo sobre los test de localización de Plantillas, la cual por defecto viene modelada y cuyo código es autogenerado. Esta ampliación sería la mostrada en la Figura 59, donde se realiza un registro configurable de las dependencias de dobles de tests. En la Figura 60, aparecería la implementación del método de utilidad para los tests sobre la localización de Plantillas, que se encarga de registrar las dependencias adicionales y devolver la instancia del *IElementLocalizationProvider*.

```
10 references | 5/5 passing | Alejandro Lozano Jiménez, 19 days ago | 1 author, 2 changes
public static void RegisterCustomDependencies(
    IServiceCollection serviceCollection,
    bool disableTemplateBuilder = false,
    bool disableTemplateLocalization = false)
{
    if (disableTemplateBuilder)
    {
        serviceCollection.AddScoped<ITemplateBuilder, DummyTemplateBuilder>();
    }
    else if (disableTemplateLocalization)
    {
        serviceCollection.AddScoped<ITemplateLocalizer, DummyTemplateLocalizer>();
    }

    serviceCollection.AddScoped<ICocktailDeploymentManager, StubDeploymentManager>();
    serviceCollection.AddScoped<IAssemblyLoadContextFactory, StubAssemblyLoadContextFactory>();
}
```

Figura 59. Método que registra dependencias personalizadas para el proyecto de tests de Reports Designer.

```
4 references | 4/4 passing | Alejandro Lozano Jiménez, 33 days ago | 1 author, 1 change
private static IElementLocalizationProvider GetElementLocalizationProviderInstance()
{
    IServiceProvider arrangeServiceProvider =
        CocktailReportsDesignerLogicTestsDependencyInjectionProvider.GetScopedServiceProvider(
            additionalRegistrations: (serviceCollection) =>
                CustomLogicTestsDependencyInjectionRegister.RegisterCustomDependencies(serviceCollection));

    return arrangeServiceProvider.GetService<IElementLocalizationProvider>();
}
```

Figura 60. Uso del método de la Figura 59 para registrar dependencias adicionales en los test sobre la localización de Plantillas.

Así, como aparece en dicha Figura 60, se puede reutilizar toda la lógica implementada mediante generación automática de código del método *GetScopedServiceProvider*, en adición al registro de dependencias adicionales como se acaba de comentar.

4.4 Resultados

Con respecto de la mejora M1 ya se han mostrado algunos ejemplos de cómo sería visualmente las Plantillas con Secciones Repetibles. Mayoritariamente, estos ejemplos eran pasos intermedios de los diferentes incrementos que se fueron haciendo; en la Figura 61 se podría observar cómo sería el caso de una Plantilla diseñada con una Sección Repetible.

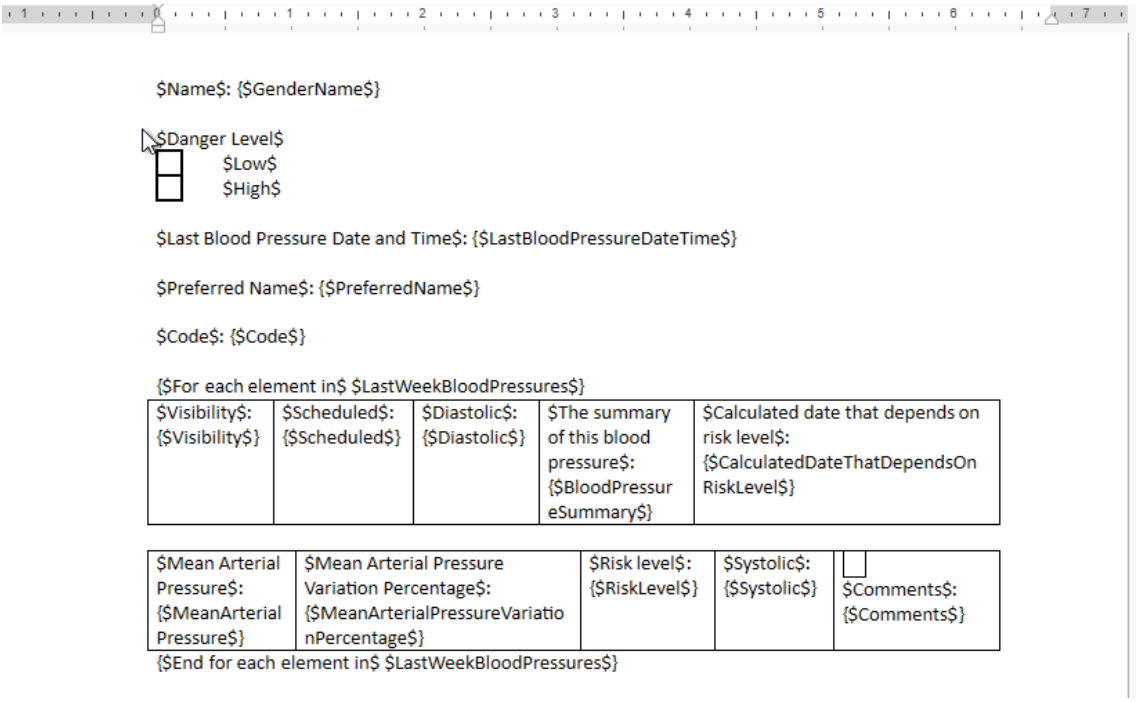

Figura 61. Ejemplo final de una Plantilla con una Sección Repetible en formato card.

Otro ejemplo, pero más sencillo y para Secciones Repetibles en formato tabla, lo podríamos observar la Figura 62.

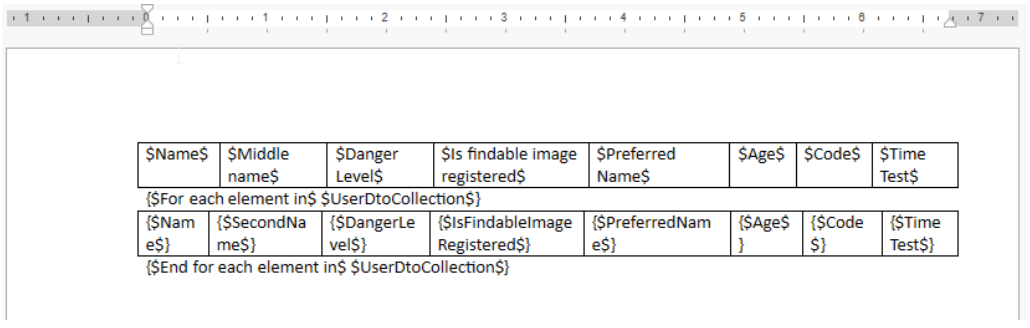

Figura 62. Ejemplo final de una Plantilla con una Sección Repetible en formato tabla.

Con respecto de la mejora M2, los resultados obtenidos son bastante simples y los esperados. Es decir, simplemente se esperaba que se pudiera mantener la integridad de los Campos insertados en el documento, de forma que las ya mencionadas operaciones intermedias nunca quedaran expuestas en el Diseñador.

Si bien con respecto del comportamiento y requisitos funcionales de la mejora no hay mucho que comentar, si es cierto que la implementación de esta solución personalizada no fue del todo satisfactoria, ya que era muy *ad hoc* y compleja. Esto estuvo presente durante toda la implementación, ya que hubo mucha prueba y error hasta entender completamente el comportamiento del histórico del Control y, de esta forma, poder adaptar la solución planteada para que trabajase con dicho histórico de forma correcta. Por ello, se decidió documentar de forma concienzuda esta problemática y la solución implementada en la wiki del equipo, con el objetivo de que sirviera de apoyo a la comprensión de ambas.

Con respecto de la mejora M3: Soporte a localización y traducción de PlantillasM3, puede observarse como partiendo de una Plantilla obtenida directamente de persistencia como es el caso de la Figura 63, pasando por su correspondiente resolución en *Reports Designer* como se aprecia en la Figura 64 y llegando a la Figura 65 donde se muestra una parte de la Plantilla localizada, se puede acabar generando un *Report* tal y como puede apreciarse en la Figura 66.

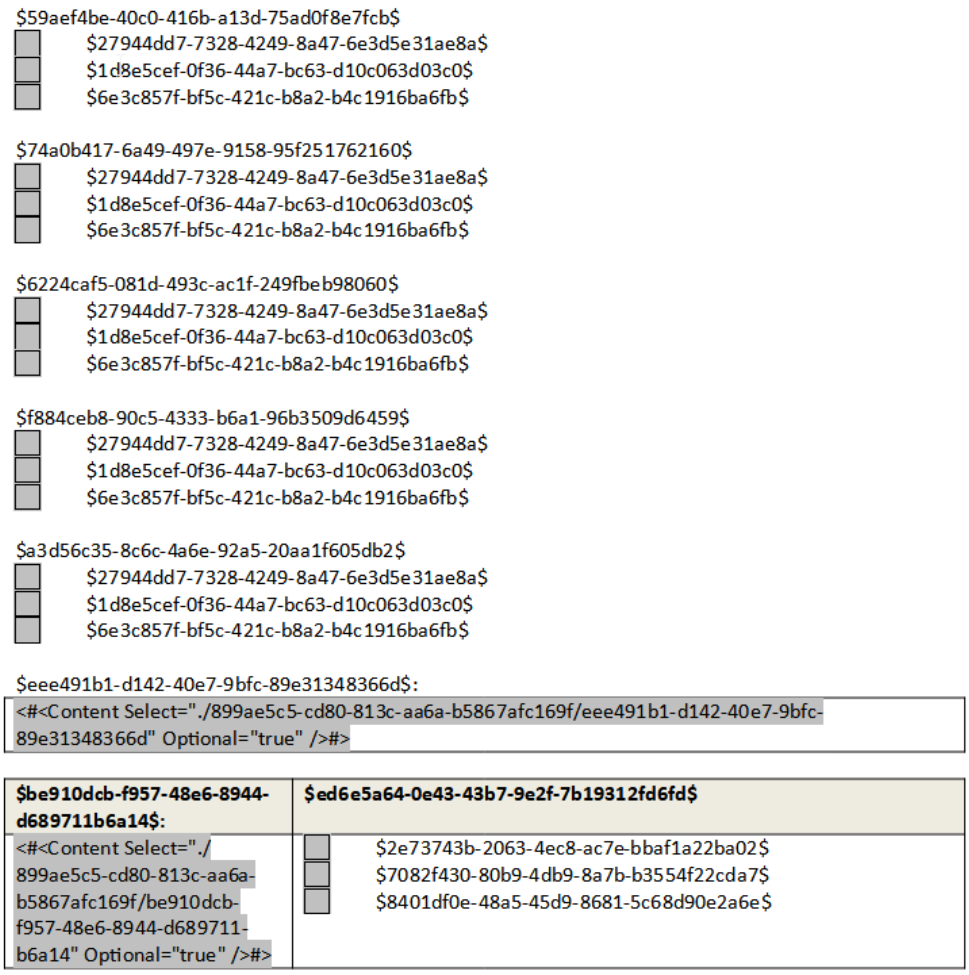

Figura 63. Ejemplo de Plantilla obtenida directamente desde persistencia.

$a3d56c35-8c6c-4a6e-92a5-20aa1f605db2$
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" NotMatch="1" Optional="true" />#>
☐ $27944dd7-7328-4249-8a47-6e3d5e31ae8a$
<#<EndConditional />#>
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" Match="1" Optional="true" />#>
☒ $27944dd7-7328-4249-8a47-6e3d5e31ae8a$
<#<EndConditional />#>
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" NotMatch="2" Optional="true" />#>
☐ $1d8e5cef-0f36-44a7-bc63-d10c063d03c0$
<#<EndConditional />#>
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" Match="2" Optional="true" />#>
☒ $1d8e5cef-0f36-44a7-bc63-d10c063d03c0$
<#<EndConditional />#>
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" NotMatch="4" Optional="true" />#>
☐ $6e3c857f-bf5c-421c-b8a2-b4c1916ba6fb$
<#<EndConditional />#>
<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/a3d56c35-8c6c-4a6e-92a5-20aa1f605-db2" Match="4" Optional="true" />#>
☒ $6e3c857f-bf5c-421c-b8a2-b4c1916ba6fb$
<#<EndConditional />#>

$eee491b1-d142-40e7-9bfc-89e31348366d$:

<#<Content Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/eee491b1-d142-40e7-9bfc-89e31348366d" Optional="true" />#>

$be910dcb-f957-48e6-8944-d689711b6a14$:	**$ed6e5a64-0e43-43b7-9e2f-7b19312fd6fd$**
<#<Content Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/be910dcb-f957-48e6-8944-d689711-b6a14" Optional="true" />#>	<#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/ed6e5a64-0e43-43b7-9e2f-7b19312fd6fd" NotMatch="1" Optional="true" />#> ☐ $2e73743b-2063-4ec8-ac7e-bbaf1a22ba02$ <#<EndConditional />#> <#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/ed6e5a64-0e43-43b7-9e2f-7b19312fd6fd" Match="1" Optional="true" />#> ☒ $2e73743b-2063-4ec8-ac7e-bbaf1a22ba02$ <#<EndConditional />#> <#<Conditional Select="./899ae5c5-cd80-813c-aa6a-b5867afc169f/ed6e5a64-0e43-43b7-9e2f-7b19312fd6fd" NotMatch="2" Optional="true" />#>

Figura 64. Ejemplo parcial de Plantilla resuelta en Reports Designer.

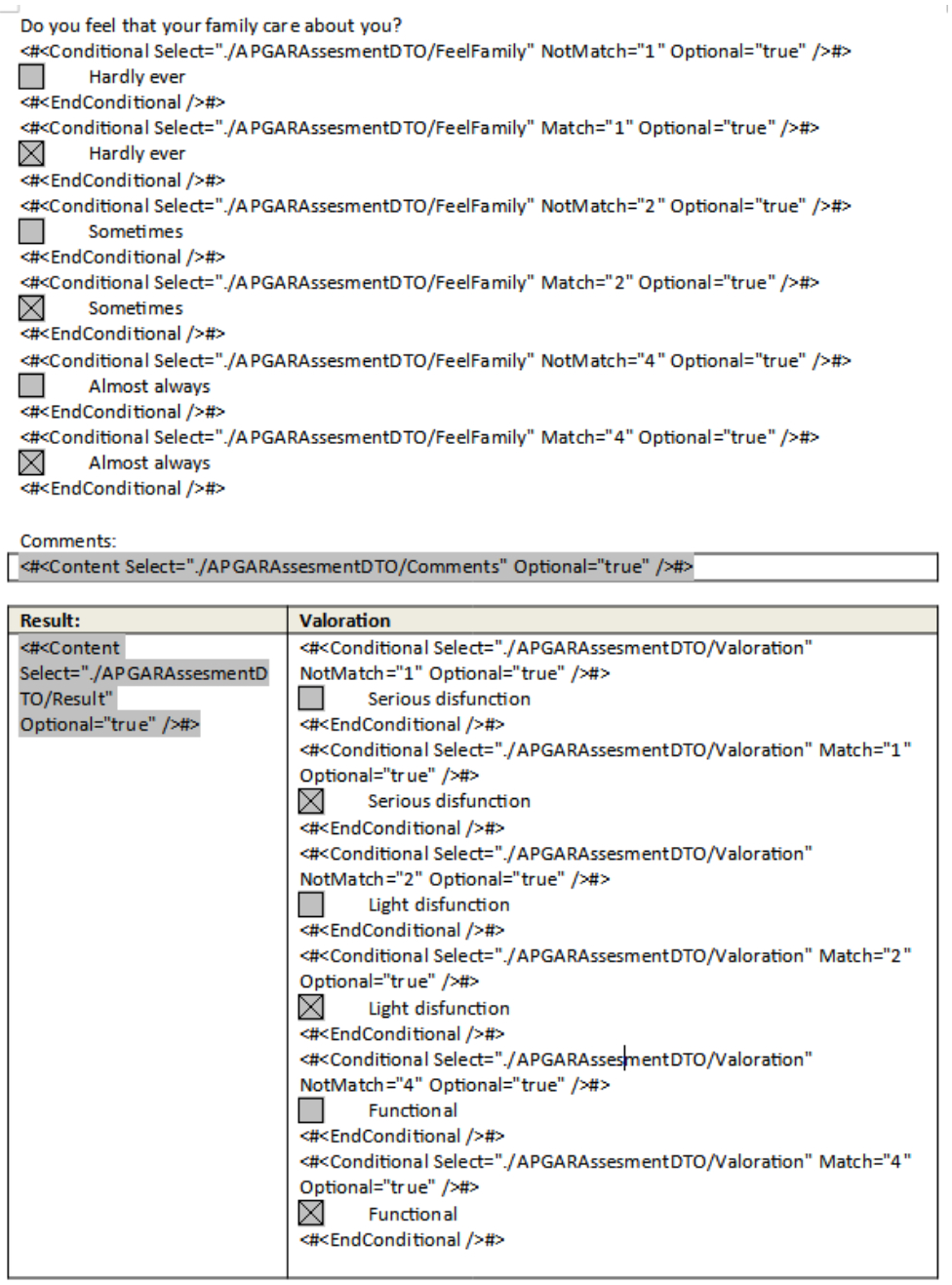

Figura 65. Ejemplo parcial de Plantilla localizada por Reports Designer.

Are you satisfied with the help that you get from your family when there's a problem?
☒ Hardly ever
☐ Sometimes
☐ Almost always

Do you all talk about the problems that have at home?
☐ Hardly ever
☒ Sometimes
☐ Almost always

Are important decisions taken by everyone in the household
☐ Hardly ever
☐ Sometimes
☒ Almost always

Are you satisfied with the time that your family and you spend together?
☐ Hardly ever
☐ Sometimes
☒ Almost always

Do you feel that your family care about you?
☒ Hardly ever
☐ Sometimes
☐ Almost always

Comments:

Assessment for testing Reports.

Result:	Valoration
5	☐ Serious disfunction ☒ Light disfunction ☐ Functional

Figura 66. Ejemplo de Report generado por Reports Engine tras la localización de su Plantilla asociada.

5. Conclusiones

En primer lugar, conviene mencionar que el conjunto de las mejoras planteadas sobre el Diseñador de Plantillas nunca estuvo cerrado. Es decir, dada la propia naturaleza del desarrollo de software y además estando el *ERP* en cuestión en su fase de concepción y preproducción, las necesidades y requisitos van variando constantemente. Es por ello por lo que, si bien había unas bases y mejoras establecidas desde el inicio del proyecto, a lo largo del desarrollo del mismo se fueron realizando modificaciones sobre esas funcionalidades o proponiendo de añadir otras nuevas. Por ello, podríamos decir que hemos cumplido satisfactoriamente nuestros objetivos: se ha implementado la gran parte de casuística considerada para las secciones repetibles para colecciones de *DTOs;* en el soporte a localización y traducción de Plantillas además de implementar también una gran parte de los casos que se habían considerado inicialmente, se ha aplicado *TDD* sin experiencia previa y con resultados bastantes satisfactorios; al respecto del ocultado de operaciones intermedias en inserciones de Campos con los comandos de deshacer y rehacer, si bien la solución no fue todo lo elegante, eficiente o legible que se esperaba conseguir, se pudo llevar a cabo de manera suficiente; y por último, también se pudieron llevar a cabo algunas menores tanto de carácter visual como funcional que permiten mejorar la experiencia de diseño de Plantillas para el presente Diseñador.

Gracias a las mejoras implementadas, se ha permitido a los analistas el diseño de Listados (mejora M1), tener una mayor consistencia y apoyo del propio Diseñador al diseño de *Reports* (mejoras M2 y M4), y permitir que una Plantilla de un *Report* pueda ser diseñada en el Diseñador de Plantillas, persistida, y posteriormente generado el propio *Report* desde el *frontend* correspondiente (mejora M3). Todas estas mejoras se encuentran desplegadas en el entorno de *stagging* de la empresa, disponibles para producción y para los usuarios internos del Diseñador de Plantillas.

Gracias a este trabajo, he podido aprender sobre diferentes tecnologías y disciplinas durante un periodo de tiempo relativamente corto. Ejemplo de ello, es la arquitectura basada en microservicios, tanto qué son conceptualmente, entendiendo sus beneficios y desventajas, así como poder poner en práctica el desarrollo y mantenimiento de dos de ellos en simultáneo. Con respecto del modelado y generación de código, si bien dadas las mejoras planteadas no se ha tenido la oportunidad de explorar en profundidad, si se ha lidiado con parte de sus consecuencias y aprovechados sus beneficios para el desarrollo de nueva funcionalidad. Además, he podido reforzar los conocimientos y experiencia en desarrollo de aplicaciones de escritorio *WPF*, aprendiendo del código existente y realizando modificaciones haciendo uso de buenas prácticas para implementar código fácilmente mantenible y extensible. Con respecto del trabajo en equipo, este proyecto ha ofrecido el contexto necesario para practicar técnicas como *pair programming* o dinámicas de revisión de código y trabajo basado en *feature branches* en *PRs*, lo que igualmente me ha permitido mejorar la capacidad para desarrollar código de calidad y aprender a trabajar bajo dichas dinámicas. De forma tangencial, también ha propiciado que adquiera conocimientos sobre *DevOps* y *CI* (*builds, pipelines)*, así como de infraestructura y manejo de productos en *Azure DevOps* [72].

Con respecto de las asignaturas de la rama de ingeniería del software, podría comentar brevemente algunas de ellas que tienen gran relación con este trabajo. En el caso de asignaturas de índole metodológico como *Proceso del Software* o *Proyecto de Ingeniería del Software*, durante este proyecto se ha podido ejercitar el desarrollo de un producto bajo un enfoque ágil; como ejemplo de algunas prácticas ágiles que se han llevado a cabo, podría mencionar que el desarrollo se hacía de manera iterativa e incremental, se preparaban las diferentes mejoras o tareas con un nivel de detalle variable dependiendo de las necesidades del momento, el alcance se reevaluaba constantemente con casi cada cambio, etc. Respecto del modelado y generación de código podría destacar *Diseño de Software dirigido por Modelos*, donde el producto sobre el que se ha desarrollado el presente proyecto se encuentra inmerso en dichas prácticas de *Model Driven Architecture (MDA)* [73]: existe un modelo de dominio sobre el cual se va trabajando, de manera que a partir de dicho modelo se autogenera un código en base a unas plantillas con partes invariables y otras que cambian en base a la información extraída del modelo. Algunas de las ventajas de esto, son la reutilización de código entre distintos microservicios y módulos, test de infraestructura reutilizables, tener código actualizado con respecto de las entidades del dominio y sus propiedades, etc. Las asignaturas de *Diseño de Software* y *Mantenimiento y Evolución del Software* también han ofrecido una base para entender las necesidades del desarrollo de código mantenible y fácilmente extensible, aplicando patrones y buenas prácticas que dirijan el desarrollo en esa dirección y la importancia de los test automatizados para la verificación del correcto funcionamiento del sistema sobre el que se trabaja.

6. Trabajo futuro

De cara a continuar con el desarrollo del Diseñador de Plantillas del presente trabajo, se propone continuar con la implementación de algunas de las mejoras propuestas que continúan en el *backlog*:

La primera de ellas sería un sistema de validaciones que aplique diferentes validaciones definidas, tanto a la hora de guardar una Plantilla abierta, como sobre todas las Plantillas de la organización. Este sistema, permitiría ir añadiendo nuevas validaciones de forma modular (el esfuerzo de añadir una nueva validación sería únicamente el necesario para definir dicha validación) y poder definir también grupos o filtros que permita activar o desactivar qué validaciones se aplican en cualquier momento. El hecho de que las validaciones puedan aplicarse sobre todas las Plantillas de la organización simultáneamente es el de evitar la necesidad de abrir las Plantillas una por una para validarlas, y evitando luego así sorpresas al generar *Reports* inválidos o vacíos.

Una mejora que los analistas encargados de modelar *Reports* habían pedido, era la de mostrar indicaciones visuales sobre qué Campos del *DTO* de datos están siendo usados en la Plantilla y cuáles no. Esto permite a los analistas, para Plantillas con *DTOs* de datos que tienen gran número de Campos, saber de un vistazo qué Campos han sido insertados ya en la Plantilla cuáles no. En un primer acercamiento, se comentó que simplemente una indicación en forma de *tick* verde o cruz roja podría ser suficiente.

Otra mejora a petición de los analistas fue la de ofrecer información extra sobre los Campos del *DTO* de datos. En concreto, la propiedad de mayor interés era la longitud máxima del Campo, ya que esto influye en cómo se plantea el diseño de ese Campo en la propia Plantilla. Otras propiedades de interés serían si es requerido o no, su tipo, ejemplos de qué valores suele tomar, etc. Una sugerencia fue que se mostrasen dichas propiedades mediante un *tooltip.*

Al respecto de los datos de entrada con los que se permite diseñar la Plantilla de un *Report,* se discutió sobre dar soporte a entidades completas del dominio, y no solo a proyecciones sobre las mismas (*DTOs)*.

Una característica que suele ser común en cualquier editor de documentos tipo *WordprocessingML* es la del uso de gráficos para representar información de manera más visual. Por ello, en el *backlog* se planteó la mejora de dar soporte a gráficos para el Diseñador de Plantillas, de forma que se permitan diseñar Plantillas con gráficos de puntos, barras o tipos similares, y que el motor de *Reports* pudiera entenderlo y generar *Reports* con dichos gráficos correctamente. Sin embargo, no se le dio mucha importancia a esta mejora ya que los propios analistas mencionaron que el número de *Reports* con gráficos no era muy grande y que había otras mejoras más prioritarias.

Otra mejora sobre la que estuvo hablando para implementar a medio plazo, fue la de dar soporte al lanzamiento de generaciones asíncronas de múltiples *Reports*. La idea, es la de que, en la *UI* del *ERP* desarrollado, se ofrezca un apartado donde poder ir marcando diferentes *Reports* y que, gestionándolo de la manera más conveniente, estos *Reports* se fueran generando en segundo plano; de esta forma, el usuario de la aplicación podría continuar su trabajo sin ser interrumpido por la generación de *Reports.*

Por último y como mejora interna, sería interesante mejorar la cobertura y calidad de las pruebas automatizadas tanto de *Reports Designer* como del propio Diseñador de Plantillas.

7. Referencias

[1] «Enterprise resource planning,» [En línea]. Available: https://en.wikipedia.org/wiki/Enterprise_resource_planning. [Último acceso: 05 Febrero 2021].

[2] L. Costello, «What is Enterprise Resource Planning (ERP)?,» Terillium, 02 Noviembre 2018. [En línea]. Available: https://terillium.com/erp-definitions/enterprise-resource-planning-erp/. [Último acceso: 05 Febrero 2021].

[3] E. (Cloud), «Brief Introduction to Enterprise Resource Planning (ERP),» SolutionDot, 24 Enero 2018. [En línea]. Available: https://solutiondots.com/blog/erp-cloud/brief-introduction-enterprise-resource-planning-erp/. [Último acceso: 20 Febrero 2021].

[4] Solver, «Permanent Link: Extending the Value of Microsoft Dynamics, Sage, SAP and Acumatica with Modern Cloud Reporting,» Solver, 04 Marzo 2020. [En línea]. Available: https://www.solverglobal.com/blog/best-cloud-reporting-tools-for-dynamics-sage-sap-acumatica/. [Último acceso: 29 Marzo 2021].

[5] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2003.

[6] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition, Pearson, 2017.

[7] M. Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2003, p. 116.

[8] Microsoft, «CultureInfo Class,» Microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=net-5.0. [Último acceso: 08 Mayo 2021].

[9] SAP, «SAP Crystal Solutions 2020,» SAP, 2021. [En línea]. Available: https://www.crystalreports.com/. [Último acceso: 29 Marzo 2021].

[10] Wikipedia, «Crystal Reports,» Wikipedia, 17 Septiembre 2020. [En línea]. Available: https://en.wikipedia.org/wiki/Crystal_Reports. [Último acceso: 29 Marzo 2021].

[11] SAP, «SAP Crystal Export Format Types,» SAP, 2020. [En línea]. Available: https://bit.ly/3qHqSXG. [Último acceso: 29 Marzo 2021].

[12] SAP, «SAP Crystal Reports 2020 data access,» SAP, 2021. [En línea]. Available: https://www.crystalreports.com/datasources/. [Último acceso: 29 Marzo 2021].

[13] SDL Trados, «TRADOStag (TTX) File Type,» SDL Trados, [En línea]. Available: http://producthelp.sdl.com/SDL%20Trados%20Studio/client_en/File_Types/TRADOStag.htm. [Último acceso: 10 Abril 2021].

[14] M. Logmans, «What's a TTX file?,» PROZ, 06 Diciembre 2007. [En línea]. Available: https://www.proz.com/forum/sdl_trados_support/91259-whats_a_ttx_file.html. [Último acceso: 10 Abril 2021].

[15] SAP, «SAP Crystal Reports Checkout,» SAP, 2021. [En línea]. Available: https://www.sapstore.com/checkout/v2. [Último acceso: 29 Marzo 2021].

[16] H. Vocke, «The Practical Test Pyramid,» martinFowlwer.com, 26 Febrero 2018. [En línea]. Available: https://martinfowler.com/articles/practical-test-pyramid.html. [Último acceso: 05 Abril 2021].

[17] B. Johnston, «Why fewer End-to-End Tests?,» Just Eat, 04 Octubre 2016. [En línea]. Available: https://tech.justeattakeaway.com/2016/10/04/why-fewer-end-to-end-tests/. [Último acceso: 05 Abril 2021].

[18] Microsoft, «Microsoft 365 para empresas,» Microsoft, [En línea]. Available: https://www.microsoft.com/es-es/microsoft-365/business. [Último acceso: 06 Abril 2021].

[19] «Model–view–viewmodel,» Wikipedia, 06 Mayo 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel. [Último acceso: 20 Mayo 2021].

[20] Acumatica, «The Benefits of Cloud Computing,» Acumatica, [En línea]. Available: https://www.acumatica.com/what-is-cloud-erp-software/. [Último acceso: 05 Mayo 2021].

[21] Acumatica, «Designing Custom Reports in Acumatica ERP,» Acumatica, [En línea]. Available: https://www.acumatica.com/cloud-erp-software/reporting-dashboards-and-data-analysis/designing-custom-reports-in-acumatica-erp/. [Último acceso: 10 Mayo 2021].

[22] Phocas, «ERP Solutions,» Phocas, [En línea]. Available: https://www.phocassoftware.com/datasources/erp-solutions. [Último acceso: 12 Mayo 2021].

[23] Software Testing Help, «10 BEST Reporting Tools in 2021 For Better Decision Making,» Software Testing Help, 30 Abril 2021. [En línea]. Available: https://www.softwaretestinghelp.com/reporting-tools. [Último acceso: 12 Mayo 2021].

[24] DBxtra, «Automatically deliver Reports on a programmed Schedule,» DBxtra, [En línea]. Available: https://dbxtra.com/report-scheduler/. [Último acceso: 12 Mayo 2021].

[25] DBxtra, «Report Web Service - Automatically Deploy Web reports,» DBxtra, [En línea]. Available: https://dbxtra.com/report-web-service/. [Último acceso: 12 Mayo 2021].

[26] Microsoft, «What is SQL Server Reporting Services (SSRS)?,» Microsoft, 05 Junio 2019. [En línea]. Available: https://docs.microsoft.com/en-us/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-ver15. [Último acceso: 13 Mayo 2021].

[27] Microsoft, «What is Power BI Report Server?,» Microsoft, 28 Mayo 2020. [En línea]. Available: https://docs.microsoft.com/en-us/power-bi/report-server/get-started#comparing-power-bi-report-server. [Último acceso: 13 Mayo 2021].

[28] Microsoft, «Reporting Services in SQL Server Data Tools (SSDT),» Microsoft, 30 Mayo 2017. [En línea]. Available: https://docs.microsoft.com/en-us/sql/reporting-

services/tools/reporting-services-in-sql-server-data-tools-ssdt?view=sql-server-ver15. [Último acceso: 13 Mayo 2021].

[29] Microsoft, «Create mobile reports with SQL Server Mobile Report Publisher,» Microsoft, 12 Junio 2018. [En línea]. Available: https://docs.microsoft.com/en-us/sql/reporting-services/mobile-reports/create-mobile-reports-with-sql-server-mobile-report-publisher?view=sql-server-ver15. [Último acceso: 13 Mayo 2021].

[30] Microsoft, «The web portal of a report server (SSRS Native Mode),» Microsoft, 31 Enero 2020. [En línea]. Available: https://docs.microsoft.com/en-us/sql/reporting-services/web-portal-ssrs-native-mode?view=sql-server-ver15. [Último acceso: 13 Mayo 2021].

[31] S. Hunter, «.NET Core is the Future of .NET,» Microsoft, 06 Mayo 2019. [En línea]. Available: https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/. [Último acceso: 08 Abril 2021].

[32] R. Lander, «Introducing .NET 5,» Microsoft, 06 Mayo 2019. [En línea]. Available: https://devblogs.microsoft.com/dotnet/introducing-net-5/. [Último acceso: 08 Abril 2021].

[33] Wikipedia, «.NET Framework,» Wikipedia, 18 Marzo 2021. [En línea]. Available: https://en.wikipedia.org/wiki/.NET_Framework. [Último acceso: 08 Abril 2021].

[34] C. De la Torre, «.NET Core, .NET Framework, Xamarin – The "WHAT and WHEN to use it",» Microsoft, 27 Junio 2016. [En línea]. Available: https://devblogs.microsoft.com/cesardelatorre/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/. [Último acceso: 06 Abril 2021].

[35] Microsoft, «What is Xamarin.Forms,» Microsoft, 28 Mayo 2020. [En línea]. Available: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms. [Último acceso: 08 Abril 2021].

[36] Microsoft, «Xamarin.Forms,» Microsoft, [En línea]. Available: https://dotnet.microsoft.com/apps/xamarin/xamarin-forms. [Último acceso: 08 Abril 2021].

[37] Microsoft, «What is Windows Presentation Foundation (WPF .NET),» Microsoft, 07 Julio 2019. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0. [Último acceso: 10 Abril 2021].

[38] Microsoft, «Binding declarations overview (WPF .NET),» Microsoft, 04 Abril 2021. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/binding-declarations-overview?view=netdesktop-5.0. [Último acceso: 23 Mayo 2021].

[39] Microsoft, «Data binding overview (WPF .NET),» Microsoft, 19 Octubre 2019. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/data-binding-overview?view=netdesktop-5.0&redirectedfrom=MSDN. [Último acceso: 10 Abril 2021].

[40] Techopedia, «Data Binding,» Techopedia, [En línea]. Available: https://www.techopedia.com/definition/15652/data-binding. [Último acceso: 10 Abril 2021].

[41] Microsoft, «Microsoft.Xaml.Behaviors.Wpf,» Microsoft, Diciembre 2019. [En línea]. Available: https://www.nuget.org/packages/Microsoft.Xaml.Behaviors.Wpf/. [Último acceso: 23 Mayo 2021].

[42] Karan, «Open Sourcing XAML Behaviors for WPF,» Microsoft, 10 Diciembre 2018. [En línea]. Available: https://devblogs.microsoft.com/dotnet/open-sourcing-xaml-behaviors-for-wpf/. [Último acceso: 23 Mayo 2021].

[43] Microsoft, «Welcome to the Open XML SDK 2.5 for Office,» Microsoft, 01 Noviembre 2017. [En línea]. Available: https://docs.microsoft.com/en-us/office/open-xml/open-xml-sdk. [Último acceso: 05 Junio 2021].

[44] OfficeDev, «Open XML SDK,» OfficeDev, [En línea]. Available: https://github.com/OfficeDev/Open-Xml-Sdk. [Último acceso: 05 Junio 2021].

[45] ECMA, «ECMA-376,» ECMA, Diciembre 2012. [En línea]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-376/. [Último acceso: 05 Junio 2021].

[46] ISO, «ISO/IEC 29500-1:2008,» ISO, Noviembre 2008. [En línea]. Available: https://www.iso.org/standard/51463.html. [Último acceso: 05 Junio 2021].

[47] Microsoft, «Structure of a WordprocessingML document (Open XML SDK),» Microsoft, 01 Noviembre 2017. [En línea]. Available: https://docs.microsoft.com/en-us/office/open-xml/structure-of-a-wordprocessingml-document. [Último acceso: 06 Junio 2021].

[48] E. White, «High Performance DOCX Generation using DocumentAssembler,» Eric White's Blog, [En línea]. Available: http://www.ericwhite.com/blog/high-performance-docx-generation-using-documentassembler/. [Último acceso: 06 Junio 2021].

[49] E. White, «Getting Started with Open-Xml-PowerTools DocumentAssembler,» Eric White's Blog, [En línea]. Available: http://www.ericwhite.com/blog/getting-started-with-open-xml-powertools-documentassembler/. [Último acceso: 06 Junio 2021].

[50] DevExpress, «WPF Rich Text Editor,» DevExpress, [En línea]. Available: https://www.devexpress.com/Products/NET/Controls/WPF/Rich_Editor/. [Último acceso: 10 Abril 2021].

[51] Microsoft, «About Domain-Specific Languages,» Microsoft, 11 Abril 2016. [En línea]. Available: https://docs.microsoft.com/en-us/visualstudio/modeling/about-domain-specific-languages?view=vs-2019. [Último acceso: 07 Abril 2021].

[52] S. Cook, G. Jones, S. Kent y A. C. Wills, Domain-Specific Development with Visual Studio DSL Tools, Addison-Wesley, 2007.

[53] S. Newman, Building Microservices, 2nd Edition, O'Reilly Media, 2021.

[54] J. Lewis y M. Fowler, «Microservices,» martinFowler.com, 25 Marzo 2014. [En línea]. Available: https://martinfowler.com/articles/microservices.html. [Último acceso: 12 Abril 2021].

[55] M. Fowler, «Microservices Guide,» 21 Agosto 2019. [En línea]. Available: https://martinfowler.com/microservices/. [Último acceso: 16 Mayo 2021].

[56] K. Beck, Test Driven Development: By Example, Addison Wesley, 2002, p. 240.

[57] Anar Corporate, «Introduction to Test Driven Development (TDD),» Anarsolutions, [En línea]. Available: https://anarsolutions.com/test-driven-development-tdd/. [Último acceso: 13 Junio 2021].

[58] Microsoft, «Create, view, and manage pull requests,» Microsoft, 28 Agosto 2020. [En línea]. Available: https://docs.microsoft.com/en-us/azure/devops/repos/git/pull-requests?view=azure-devops. [Último acceso: 30 Marzo 2021].

[59] Microsoft, «ParagraphProperties Class,» Microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/api/documentformat.openxml.wordprocessing.paragraphproperties?view=openxml-2.8.1. [Último acceso: 31 Mayo 2021].

[60] Microsoft, «BiDi Class,» Microsoft, [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/api/documentformat.openxml.wordprocessing.bidi?view=openxml-2.8.1. [Último acceso: 31 Mayo 2021].

[61] A. L. Jiménez, «Validation error - Unexpected element 'BiDi' in WordprocessingDocument,» GitHub, 2020 Diciembre 23. [En línea]. Available: https://github.com/OfficeDev/Open-XML-SDK/issues/830. [Último acceso: 31 Mayo 2021].

[62] FileFormat, «DOCX - File Format Specifications,» FileFormat, [En línea]. Available: https://docs.fileformat.com/word-processing/docx/#file-format-specifications. [Último acceso: 01 Junio 2021].

[63] Microsoft, «Structure of an Open XML Package,» Microsoft, 01 Noviembre 2017. [En línea]. Available: https://docs.microsoft.com/en-us/office/open-xml/about-the-open-xml-sdk#structure-of-an-open-xml-package. [Último acceso: 06 Junio 2021].

[64] DevExpress, «How to manage the undo actions queue for RFTEditor,» DevExpress, 11 Mayo 2015. [En línea]. Available: https://supportcenter.devexpress.com/ticket/details/t240247/how-to-manage-the-undo-actions-queue-for-rfteditor. [Último acceso: 06 Junio 2021].

[65] DevExpress, «SpreadsheetControl - How to undo/redo several document changes at once,» DevExpress, 24 Julio 2019. [En línea]. Available: https://supportcenter.devexpress.com/ticket/details/t800928/spreadsheetcontrol-how-to-undo-redo-several-document-changes-at-once. [Último acceso: 06 Junio 2021].

[66] DevExpress, «More than one step in Undo,» DevExpress, 14 Septiembre 2016. [En línea]. Available: https://supportcenter.devexpress.com/ticket/details/t427148/more-than-one-step-in-undo. [Último acceso: 06 Junio 2021].

[67] DevExpress, «Adding custom history item to DocumentModel.History,» DevExpress, 14 Abril 2016. [En línea]. Available: https://supportcenter.devexpress.com/ticket/details/t367670/adding-custom-history-item-to-documentmodel-history. [Último acceso: 06 Junio 2021].

[68] Microsoft, «Understanding System.Runtime.Loader.AssemblyLoadContext,» Microsoft, 09 Agosto 2019. [En línea]. Available: https://docs.microsoft.com/en-

us/dotnet/core/dependency-loading/understanding-assemblyloadcontext. [Último acceso: 09 Junio 2021].

[69] Google, «RFC 4646 - Tags for Identifying Languages,» Google, Septiembre 2006. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc4646. [Último acceso: 09 Junio 2021].

[70] Microsoft, «Dependency injection in .NET,» Microsoft, 12 Abril 2021. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection. [Último acceso: 24 Mayo 2021].

[71] G. Meszaros, «Test Double,» xUnit Patterns, 09 Febrero 2011. [En línea]. Available: http://xunitpatterns.com/Test%20Double.html. [Último acceso: 18 Junio 2021].

[72] Microsoft, «Azure Repos,» Microsoft, [En línea]. Available: https://azure.microsoft.com/en-us/services/devops/repos/. [Último acceso: 27 Mayo 2021].

[73] Wikipedia, «Model-driven architecture,» Wikipedia, 29 Abril 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Model-driven_architecture. [Último acceso: 27 Mayo 2021].

[74] Microsoft, «XAML overview,» Microsoft, 07 Julio 2018. [En línea]. Available: https://docs.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview. [Último acceso: 10 Abril 2021].

[75] «About,» [En línea]. Available: https://www.eclipse.org/birt/about/. [Último acceso: 12 Mayo 2021].

[76] P. L. Torres, «El Backlog: el contenedor del trabajo pendiente,» Agilismo At Work, 25 Mayo 2014. [En línea]. Available: http://agilismoatwork.blogspot.com/2014/05/backlog-todo-el-trabajo-pendiente-del.html. [Último acceso: 18 Mayo 2021].

[77] P. L. Torres, «Organización ágil del trabajo: Parte I - Líneas de Trabajo,» Agilismo At Work, 02 Mayo 2019. [En línea]. Available: https://agilismoatwork.blogspot.com/2019/05/organizacion-agil-del-trabajo.html. [Último acceso: 18 Mayo 2021].

[78] E. White, «DocumentAssembler GettingStarted,» Eric White, 07 Septiembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=LqZwCQbK4qU. [Último acceso: 01 Junio 2021].

[79] Microsoft, «Introduction to ASP.NET Core,» Microsoft, 17 Abril 2020. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0. [Último acceso: 17 Junio 2021].

[80] Microsoft, «Introduction to ASP.NET Core SignalR,» Microsoft, 27 Noviembre 2019. [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0. [Último acceso: 17 Junio 2021].

[81] M. Lipski, «Test Doubles — Fakes, Mocks and Stubs.,» Pragmatists, 30 Marzo 2017. [En línea]. Available: https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da. [Último acceso: 18 Junio 2021].

[82] G. Meszaros, «Fake Object,» xUnit Patterns, 09 Febrero 2011. [En línea]. Available: http://xunitpatterns.com/Fake%20Object.html. [Último acceso: 18 Junio 2021].