UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Visualización y análisis de métricas de desarrollo y mantenimiento en productos software

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Silvia Martos Martínez

Tutor: Patricio Letelier Torres
José Hilario Canos Cerda

Curso 2018-2019

Resum

Els productes programari reeixit requereixen manteniment, si el programari s'usa intensament, apareixeran necessitats d'extensió, adaptació, integració, etc. Quan a més dits productes tenen una certa envergadura, els equips de desenvolupadors són grans i la quantitat de canvis és alta, tot això porta associada la gestió de molta informació del procés de desenvolupament (canvis, especificacions, proves, fallades, etc). En aquest context sorgeix la necessitat de disposar de mecanismes per a la visualització de dades que faciliten la seua anàlisi. Existeix un ampli ventall de tècniques i tipus de gràfics que es poden utilitzar (Gràfics de barres, Gràfics Circulars, Gràfics de Bambolla, Infografies, etc.). El propòsit d'aquestes visualitzacions és que els desenvolupadors puguen comprendre i explotar aqueixes dades per a prendre decisions sobre el manteniment del producte, per exemple, quines parts del producte presenten més canvis, quines són les taxes de fallades, quin és el grau de testing que s'està aplicant, etc.

L'objectiu d'aquest Treball Final de Grau (TFG) és avaluar tècniques per a visualització de dades i aplicar-les en el context de manteniment de productes programari de gran envergadura. Aquest TFG s'emmarca en una pràctica d'empresa realitzada per l'autora. L'empresa en la qual es realitza aquesta pràctica comercialitza un Sistema de Planificació de Recursos Empresarials (ERP) orientat al sector soci sanitari. Aquest producte té una gran quantitat de funcionalitats i està implantat en més de 1500 clients. L'equip de desenvolupament el formen quasi 30 persones. Els resultats d'aquest TFG podran donar suport a la presa de decisions pel que fa al manteniment del producte i la millora contínua del procés de desenvolupament.

Paraules clau: Gestió de un procés de desenvolupament, visualització de dades, big data, anàlisi estadistic

Resumen

Los productos software exitosos requieren mantenimiento, si el software se usa intensamente, aparecerán necesidades de extensión, adaptación, integración, etc. Cuando además dichos productos tienen una cierta envergadura, los equipos de desarrolladores son grandes y la cantidad de cambios es alta, todo esto lleva asociada la gestión de mucha información del proceso de desarrollo (cambios, especificaciones, pruebas, fallos, etc). En este contexto surge la necesidad de disponer de mecanismos para la visualización de datos que faciliten su análisis. Existe un amplio abanico de técnicas y tipos de gráficos que se pueden utilizar (Gráficos de barras, Gráficos Circulares, Gráficos de Burbuja, Infografías, etc.). El propósito de estas visualizaciones es que los desarrolladores puedan comprender y explotar esos datos para tomar decisiones sobre el mantenimiento del producto, por ejemplo, qué partes del producto presentan más cambios, cuáles son las tasas de fallos, cuál es el grado de *testing* que se está aplicando, etc.

El objetivo de este Trabajo Final de Grado (TFG) es evaluar técnicas para visualización de datos y aplicarlas en el contexto de mantenimiento de productos software de gran envergadura. Este TFG se enmarca en una práctica de empresa realizada por la autora. La empresa en la que se realiza dicha práctica comercializa un Sistema de Planificación de Recursos Empresariales (ERP) orientado al sector socio sanitario. Este producto tiene una gran cantidad de funcionalidades y está implantado en más de 1500 clientes. El equipo de desarrollo lo forman casi 30 personas. Los resultados de este TFG podrán apoyar la toma de decisiones en lo que respecta al mantenimiento del producto y la mejora continua del proceso de desarrollo.

Palabras clave: Gestión de un proceso software, visualización de datos, big data, análisis estadístico

Abstract

Successful software products require maintenance, if the software is used intensively, needs of extension, adaptation, integration, etc. will appear. When these products also have a certain size, the teams of developers are large and the amount of changes is high, all this is associated with the management of a lot of information of the development process (changes, specifications, tests, failures, etc). In this context, there is a need for mechanisms for the visualization of data that facilitate its analysis. There is a wide range of techniques and chart types that can be used (Bar Charts, Circular Charts, Bubble Charts, Infographics, etc.). The purpose of these visualizations is for developers to understand and exploit that data to make decisions about product maintenance, for example, which parts of the product have the most changes, what are the failure rates, what is the degree of testing being applied, etc.

The objective of this Final Grade Work (TFG) is to evaluate techniques for data visualization and to apply them in the context of maintenance of large software products. This TFG is part of a company practice carried out by the author. The company in which this practice is carried out markets an Enterprise Resource Planning System (ERP) oriented to the healthcare partner sector. This product has a lot of functionalities and is implemented in more than 1500 customers. The development team is made up of almost 30 people. The results of this TFG will be able to support decision-making regarding product maintenance and continuous improvement of the development process.

Key words: Management of a software process, visualization of data, big data, statistical analysis

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El mantenimiento del software es una de las fases del ciclo de vida de desarrollo de sistemas y no solo engloba la corrección de defectos. Un 80 % del esfuerzo de mantenimiento es invertido en mejoras y nuevas funcionalidades [4] .

El propósito del mantenimiento es preservar el valor del software sobre el tiempo alargando su vida util y retrasando el costo económico que supondría una migración total hacia una nueva aplicación. Para ello, a lo largo de la vida útil de una aplicación, esta puede necesitar modificaciones por distintas razones:

- Para detectar problemas que puedan surgir en el futuro (mantenimiento preventivo).
- Corregir defectos encontrados y que originan un comportamiento distinto al esperado (mantenimiento correctivo).
- Cambiar el entorno de uso de la aplicación (mantenimiento adaptativo).
- Agregar nuevas funcionalidades o características (mantenimiento perfectivo).

Los sucesivos cambios producidos por el mantenimiento hacen que el código sea más difícil de modificar. Según Sommerville : "Cualquier cambio conlleva la corrupción de la estructura del software y, a mayor corrupción, la estructura del programa se torna menos comprensible y mas difícil de modificar" [5]. Y a esto le añadimos que todo cambio lleva un costo asociado. Para hacer un cambio es necesario comprender el software y entender lo que se necesita, a continuación modificar el programa y finalmente realizar pruebas para validar los cambios.

Es importante tener indicadores que nos ayuden en el análisis del producto. Poder visualizar toda la información almacenada en cada uno de estos cambios mediante un abanico de técnicas y gráficos que puedan ayudar a los desarrolladores a comprender el alcance de sus acciones (análisis de impacto [2]), ya sean preventivas, correctivas, adaptativas y/o perfectivas.

Por otra parte, también dicha visualización sirve para tomar decisiones de planificación de cambios, de automatizar partes del producto, priorizar acciones de mejora, etc.

1.1 Motivación

La motivación principal de este trabajo es ayudar a gestionar un producto software de gran envergadura, y con ello reducir el tiempo dedicado por el equipo de desarrollo en sus tareas cotidianas de mantenimiento. Con equipo de desarrollo, no solo nos referimos a los programadores que implementan los cambios, sino también a los analistas y a los *testers*. Los analistas y *testers* deben definir y probar cambios en productos software que van evolucionando, haciéndose más grandes y más complejos, para ello gestionar la información de requisitos es fundamental.

Segun MCclure [6] , el tiempo empleado en las tareas de mantenimiento se distribuye de la siguiente forma:

- 18 % Estudiar peticiones.
- 6 % Estudiar documentación.
- 13 % Estudiar código.
- 19 % Implementar cambio.
- 28 % Realizar pruebas.
- 6 % Actualizar documentación.

Todo lo que ayude a reducir este tiempo contribuye también a reducir los costos asociados al cambio.

En las metodologías ágiles después de la primera entrega del producto se puede considerar que las posteriores modificaciones se engloban más dentro del ciclo del mantenimiento del producto que del desarrollo del mismo.

El presente trabajo se desarrolla en el contexto de una empresa que se dedica a la comercialización y mantenimiento de un producto software de gestión para el sector socio-sanitario. Dicho producto se empezó a desarrollar en el año 1997 y como cabe esperar, a día de hoy, el software ya ha alcanzado un tamaño considerable y hay un equipo de más de 30 personas trabajando a diario en su mantenimiento.

El proceso de desarrollo utilizado en la empresa es iterativo e incremental, aproximadamente se publican 12 versiones al año, una por cada mes. Cada una de estas iteraciones incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación.

En el desarrollo del software utilizan metodologías ágiles con una orientación basada en pruebas de aceptación *(Test-Driven Requeriment Engineering (TDRE))*[1]. En este enfoque se consideran las pruebas como un proceso paralelo al análisis y el desarrollo y no como una fase aislada al final del proceso. El planteamiento es que hasta que no se tengan definidas las pruebas de aceptación que ese requisito debe satisfacer no se debe empezar a implementar.

Una prueba de aceptación (PA) es un escenario de utilización del sistema y el comportamiento que de él se espera, visto desde la perspectiva del usuario final. Las PA constituyen el criterio de éxito en cuanto a la implementación de un requisito del sistema.

Se le llama incidencia a la unidad de trabajo del proceso. Ésta pasa a través de las actividades siendo atendida por los agentes responsables de dichas actividades hasta finalizar su ciclo de vida en un cierre de versión.

El programador implementa el código con la idea de superar las PA de forma incremental, hasta completar toda la funcionalidad requerida y pasar todas las pruebas de aceptación asociadas a la incidencia.

Las pruebas de sistema (PS) son instanciaciones concretas de las PA y son aplicadas por los *testers* para comprobar el éxito/fracaso de la implementación de un requisito.

A Junio de 2019, se tiene un total de 17.792 pruebas de aceptación y 3.975 pruebas de sistema definidas en los aproximadamente 3.254 nodos de los que consta el programa de estudio.

La empresa cuenta con un herramienta interna para gestionar el producto. En dicha herramienta se especifican los requisitos del programa y mediante un árbol de nodos se agrupan las distintas funcionalidades pudiendo visualizar la estructura global de la aplicación y ver que nodos tienen más pruebas de aceptación o acumulan mayor cantidad de incidencias de fallo.

Poder visualizar esta información contenida en la herramienta interna de la empresa de una manera gráfica y con ello ayudar a responder cuestiones tales como: donde se acumulan los fallos, que partes del programa han sufrido más cambios o donde se encuentran definidas más pruebas de sistema, son ejemplos de mejoras que se abordaran con este TFG.

1.2 Objetivos

El desarrollo del presente trabajo tiene como objetivo abordar la problemática de visualizar los datos de un producto software cuando este tiene gran envergadura. Para ello se hará uso de herramientas ya existentes en el mercado.

Los objetivos específicos son:

- Estudiar técnicas y gráficos dinámicos para visualizar la estructura de un producto software.
- Ilustrar como se puede ayudar en la toma de decisiones y la mejora del proceso apoyándose en las visualizaciones estudiadas.

1.3 Estructura de la memoria

La memoria está organizada en un total de nueve capítulos.

En el capítulo 1 se incluye una introducción a la motivación del trabajo y presenta los objetivos.

En el capítulo 2 se presentan las herramientas de las que se dispone actualmente en el contexto de la empresa de estudio y que están relacionadas con el análisis estadístico y se plantea el problema que se resolverá con el presente trabajo.

En el capítulo 3 se evalúan y comparan las distintas herramientas de gestión de producto existentes en el mercado haciendo especial hincapié en como gestionan la estructura del producto y si proporcionan gestión de pruebas.

En el capítulo 4 se trabaja en la corrección de los cálculos que nos proporcionan los instrumentos actuales de los que dispone la empresa.

En el capítulo 5 se hace una breve introducción a las distintos artefactos de visualización que se van a utilizar para presentar informes.

En el capítulo 6 se describen las transformaciones que se han tenido que ir implantando para poder utilizar los datos en las herramientas de visualización.

En el capítulo 7 se presentan las distintas gráficas generadas con los distintos artefactos vistos anteriormente.

En el capítulo 8 se introduce una comparativa sobre las herramientas resaltando ventajas e inconvenientes de cada una.

En el capítulo 9 se tratan las conclusiones y el trabajo futuro que surge a partir de este trabajo.

CAPÍTULO 2
Definición del problema

En este apartado se va a abordar la problemática de mantener un producto software que se empezó a desarrollar hace mas de dos décadas y que año tras año ha ido creciendo para satisfacer las necesidades de sus consumidores. También se verán los mecanismos de análisis de datos de los que dispone actualmente la empresa y la problemática de aplicarlos para sacar informes sobre las partes del producto que más se están usando o donde se introducen más fallos u otras cuestiones de interés para tomar mejores decisiones de mantenimiento.

2.1 Sistema de Ayuda al Proceso de Incidencias (SAPI)

SAPI es una herramienta interna de la empresa que surge como necesidad para coordinar el trabajo entre los distintos agentes involucrados en el desarrollo de un requisito. Permite y facilita la gestión y mantenimiento del producto, desde el *backlog* hasta el cierre de versión mediante el uso de flujos de trabajo ágiles. También permite la comunicación entre los distintos usuarios o agentes mediante el envío de peticiones a nivel global o asociadas a una incidencia o incluso a una prueba de aceptación concreta.

SAPI tiene diversos módulos principales y voy a nombrar los mas importantes para luego centrar la atención en el que nos interesa para este TFG:

- Planificador de Versiones : Permite gestionar la información sobre las versiones, ver el trabajo de cada agente en una versión concreta, administrar los flujos de trabajo...

- Planificador Personal : Permite visualizar un listado de todas las actividades que un agente tiene asignadas y el estado en el que se encuentra dicha actividad.

- Gestor de Requisitos (GR): Es donde se establece la estructura del producto y donde se especifican los requisitos del programa. Se organiza mediante un árbol de nodos y dentro de cada nodo se agrupan las pruebas de aceptación dependiendo de la funcionalidad a la que afectan. Se hará especial hincapié en él puesto que es donde está la vista estadística y de donde parte este TFG para obtener las posteriores visualizaciones.

2.1.1. Vista Estadística del GR

Como se comentaba anteriormente, en el GR se define la estructura del producto. Como se puede observar en la Figura 2.1, está estructurado en nodos con relaciones padre-hijo, y dentro de cada nodo están las pruebas de aceptación que como ya se ha remarcado antes son las que especifican los requisitos del producto.

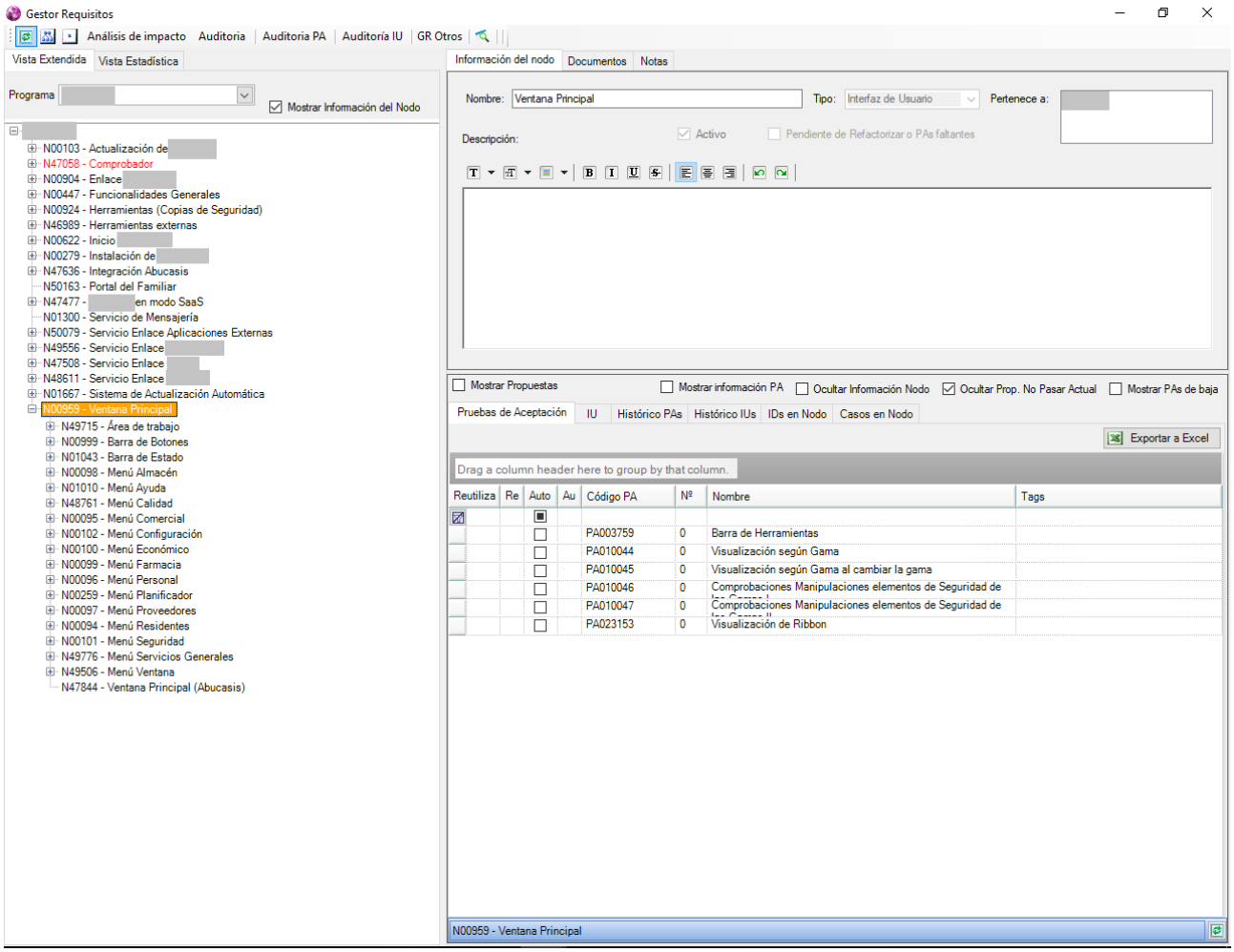


Figura 2.1: Vista Estadística del GR

En la Figura 2.1 se puede ver a su izquierda seleccionado el nodo "Ventana Principal" que por una parte es hijo del nodo principal del programa y por otra parte es padre de todos los nodos que cuelgan de él.

A la derecha de dicha Figura se puede ver las características de dicho nodo. En concreto en esta imagen se muestran las pruebas de aceptación que debe cumplir, pero se podrían mostrar otras pestañas tales como: las interfaces de usuario que le afectan, las incidencias que tiene o las pruebas de sistema que se han definido para comprobar el cumplimiento de las PA.

En la Figura 2.2 se muestra la pestaña "Automatización" de la vista estadística del GR. El primer propósito de este TFG ha sido mejorar los datos calculados que se muestran al lado de cada nodo.

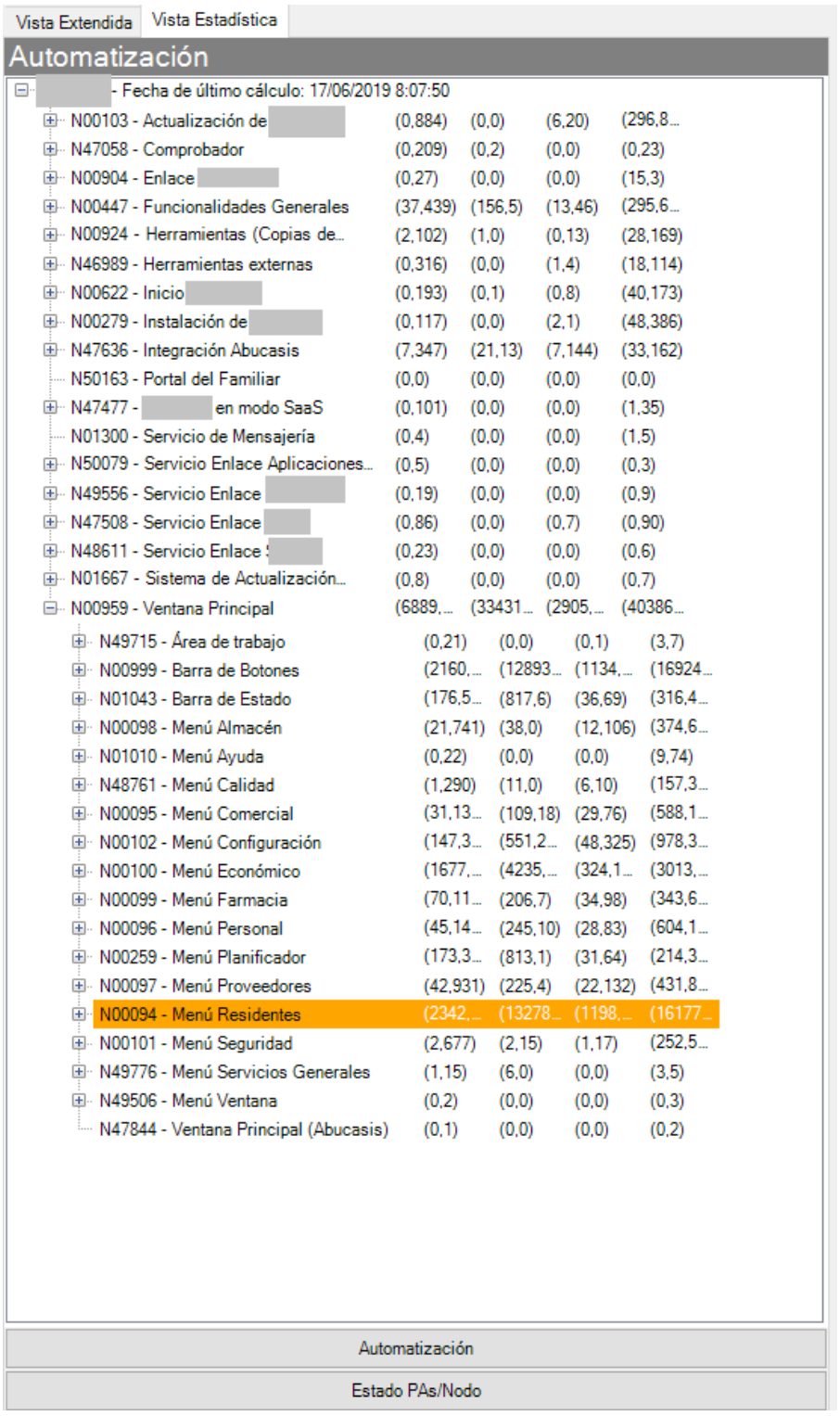


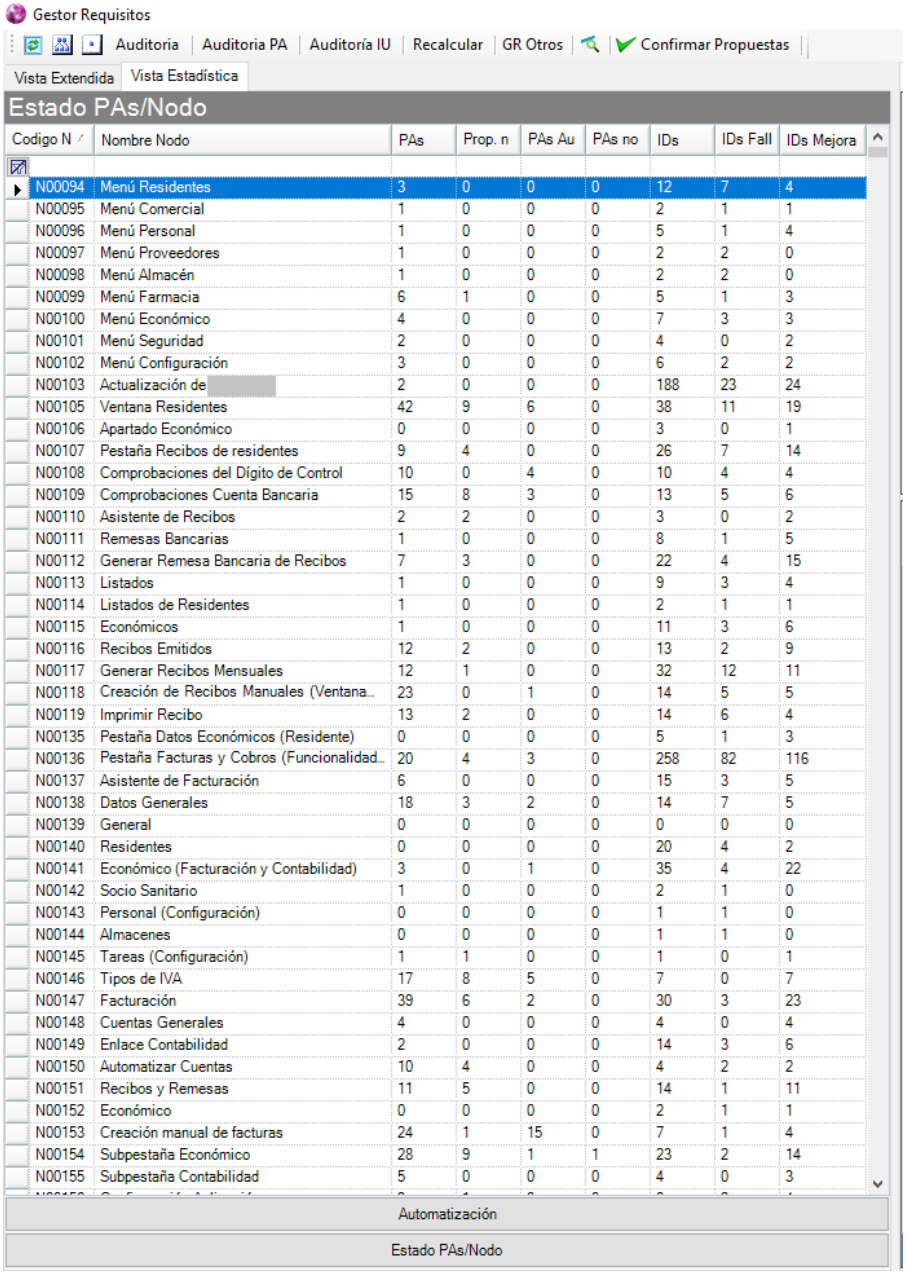
Figura 2.2: Pestaña Automatización del GR

Los valores de izquierda a derecha corresponden a : (PA automatizadas, PA totales), (PS automatizadas, PS totales), (Fallos pendientes, Fallos cerrados), (Incidencias de tipo fallo, Otras incidencias).

En estos cálculos que se muestran no se tuvo en cuenta la complejidad del software que se está analizando y para obtener los valores de los distintos nodos no se contempló la posibilidad de que pudieran haber ciclos en las estructuras padre-hijo de los nodos. Eso ha ocasionado que los datos que se exponen no son correctos y no sirven como partida para analizar nuestro producto.

En el Capítulo 4 se mostrará, ilustrándolo con un caso teórico, cómo se han mejorado estos cálculos pero ahora vamos a continuar analizando las características que nos ofrece el GR.

Otra pestaña interesante dentro de esta vista estadística es "Estado PAs/Nodo", en ella podemos ver todos los datos dentro de un *grid*, el cual permite filtrar u ordenar de acuerdo a una columna concreta. Tiene una característica importante y es que en esta vista solo se ofrecen los datos concretos de ese nodo.



Codigo N	Nombre Nodo	PAs	Prop. n	PAs Au	PAs no	IDs	IDs Fall	IDs Mejora
N00094	Menú Residentes	3	0	0	0	12	7	4
N00095	Menú Comercial	1	0	0	0	2	1	1
N00096	Menú Personal	1	0	0	0	5	1	4
N00097	Menú Proveedores	1	0	0	0	2	2	0
N00098	Menú Almacén	1	0	0	0	2	2	0
N00099	Menú Farmacia	6	1	0	0	5	1	3
N00100	Menú Económico	4	0	0	0	7	3	3
N00101	Menú Seguridad	2	0	0	0	4	0	2
N00102	Menú Configuración	3	0	0	0	6	2	2
N00103	Actualización de	2	0	0	0	188	23	24
N00105	Ventana Residentes	42	9	6	0	38	11	19
N00106	Apartado Económico	0	0	0	0	3	0	1
N00107	Pestaña Recibos de residentes	9	4	0	0	26	7	14
N00108	Comprobaciones del Dígito de Control	10	0	4	0	10	4	4
N00109	Comprobaciones Cuenta Bancaria	15	8	3	0	13	5	6
N00110	Asistente de Recibos	2	2	0	0	3	0	2
N00111	Remesas Bancarias	1	0	0	0	8	1	5
N00112	Generar Remesa Bancaria de Recibos	7	3	0	0	22	4	15
N00113	Listados	1	0	0	0	9	3	4
N00114	Listados de Residentes	1	0	0	0	2	1	1
N00115	Económicos	1	0	0	0	11	3	6
N00116	Recibos Emitidos	12	2	0	0	13	2	9
N00117	Generar Recibos Mensuales	12	1	0	0	32	12	11
N00118	Creación de Recibos Manuales (Ventana...	23	0	1	0	14	5	5
N00119	Imprimir Recibo	13	2	0	0	14	6	4
N00135	Pestaña Datos Económicos (Residente)	0	0	0	0	5	1	3
N00136	Pestaña Facturas y Cobros (Funcionalidad...	20	4	3	0	258	82	116
N00137	Asistente de Facturación	6	0	0	0	15	3	5
N00138	Datos Generales	18	3	2	0	14	7	5
N00139	General	0	0	0	0	0	0	0
N00140	Residentes	0	0	0	0	20	4	2
N00141	Económico (Facturación y Contabilidad)	3	0	1	0	35	4	22
N00142	Socio Sanitario	1	0	0	0	2	1	0
N00143	Personal (Configuración)	0	0	0	0	1	1	0
N00144	Almacenes	0	0	0	0	1	1	0
N00145	Tareas (Configuración)	1	1	0	0	1	0	1
N00146	Tipos de IVA	17	8	5	0	7	0	7
N00147	Facturación	39	6	2	0	30	3	23
N00148	Cuentas Generales	4	0	0	0	4	0	4
N00149	Enlace Contabilidad	2	0	0	0	14	3	6
N00150	Automatizar Cuentas	10	4	0	0	4	2	2
N00151	Recibos y Remesas	11	5	0	0	14	1	11
N00152	Económico	0	0	0	0	2	1	1
N00153	Creación manual de facturas	24	1	15	0	7	1	4
N00154	Subpestaña Económico	28	9	1	1	23	2	14
N00155	Subpestaña Contabilidad	5	0	0	0	4	0	3

Figura 2.3: Vista Estado PAs/Nodo

Cabe resaltar que estos cálculos que aparecen en la Figura 2.3 si que son correctos puesto que no se tiene en cuenta la herencia para obtenerlos.

Esta información a nivel de nodo que se muestra en el *grid* de la Figura 2.3 es la que posteriormente aparecerá representada en los distintos gráficos para ayudar a interpretar su significado de un modo más visual.

Aquí tenemos una lista de los nodos con, por ejemplo, el número de incidencias (ID) que le afectan. Los nodos lógicamente se pueden ordenar de acuerdo al valor de esa columna, de mayor número

de ID a menor número de éstas pero ese valor no deja de ser un número en una celda. Visualizar esta información de un modo gráfico, con el cual, de una sola mirada, se pueda sacar conocimiento de esos datos, es uno de los objetivos principales de este TFG.

2.1.2. Uso del SAPI

SAPI es la herramienta más utilizada por los analistas de la empresa puesto que en ella se definen los requisitos del producto.

Pero no solo los analistas hacen uso de esta herramienta, todos los agentes involucrados en una incidencia tienen acceso a través del SAPI a la información necesaria para llevar adelante los cambios necesarios que satisfagan las necesidades del cliente.

Entre otras características de la herramienta hay una pestaña documentación dentro de cada incidencia donde aparte del documento de análisis puede haber otros archivos importantes para la implementación.

El programador toma decisiones a nivel de diseño y en el nivel de detalle que le resulte adecuado con el propósito de identificar alternativas de implementación y establecer las estimaciones de esfuerzo asociadas. Esto se plasma en un documento de diseño que también se adjunta a la incidencia en el apartado de documentación antes mencionado.

Los programadores hacen uso también de una pestaña llamada programación donde el analista ha asociado pruebas de aceptación que dicho requisito debe cumplir. Estas pruebas pueden ser nuevas propuestas o pruebas de regresión que comprueban que después del cambio todo sigue funcionando correctamente.

También los *testers* hacen uso de SAPI. Existe una pestaña de testeo donde también ellos marcan las pruebas como *OK/KO* después de que el programador haya terminado de implementar el cambio.

Con esto se pretende resaltar que SAPI es una herramienta muy importante en el día a día de la empresa y se le da un uso intensivo en todo el departamento de desarrollo.

2.2 Problemática actual

Como se ha podido observar existen herramientas ya implementadas dentro de la empresa para gestionar la información sobre la aplicación.

A nivel interno se dispone de SAPI donde se pueden ver todos los nodos del programa, el número de PA que tiene cada nodo, las funcionalidades que cubre, las incidencias que ha tenido (mejora, fallo, nuevo requisito, otras). Pero, ¿de que sirve tanta información si no es correcta, no esta bien estructurada y no se puede sacar conocimiento de ella?

Según un estudio reciente llevado a cabo por IBM, el 88 % de los datos almacenados en las empresas no se utilizan o no son tenidos en cuenta a la hora de tomar decisiones [3]. De esta forma, los datos que se tienen del producto, no se aprovechan del todo por su complejidad y su poco apropiada visualización.

Aquí es donde entra en acción el *Big Data* [8]. Hay muchas definiciones sobre que es y para que sirve pero habría que resaltar el concepto principal que no es otro que dar valor a los datos, concretamente poder seleccionar lo que es importante y deshechar lo que no aporta conocimiento inmediato sobre el estado del producto.

Implantar el *Big Data* en una empresa es un proceso complejo y que requiere tiempo, dedicación e implicación de todo el equipo de trabajo. Este TFG mostrará las posibilidades que ofrecen nuestros datos mediante visualizaciones atractivas de la información que se haya oculta en la gran cantidad de datos que se tienen almacenados.

En este TFG no se pretende implantar el *Big Data* sino estudiar alternativas para su presentación que ayuden a analizar y sacar información de los datos, específicamente en cuanto a los asociados con la visualización de la estructura del producto.

CAPÍTULO 3

Estado del arte. Gestión de un producto software

La gestión de un producto software no solo implica la gestión de requisitos sino también poder explotar la información asociada a la estructura del producto. En el capítulo anterior hemos presentado la herramienta de gestión interna de la empresa en la que se desarrolla este TFG, SAPI, la cual aparte de estar orientada a la especificación de requisitos también engloba otras fases tales como el desarrollo y el mantenimiento del producto a través de la estructura definida en el GR.

En la actualidad existe una gran variedad de herramientas en el mercado para poder gestionar los requisitos de un producto software pero además sería necesario que la misma herramienta se encargará también de la gestión de la estructura del producto.

3.1 Comparativa entre las herramientas

A continuación, se presentarán algunas de las herramientas más populares que se han estudiado a la hora de buscar una herramienta de gestión de producto que permitiera partir de la estructura subyacente para generar visualizaciones: IBM Rational Doors [1], Visure Requirements [2], Jama Product Development [3] y RequisitePro [4].

El estudio de estas herramientas incluirá una breve descripción de ellas y se centrará en como se organizan (estructura del producto) y en si registran información de pruebas, fallos, etc.

3.1.1. IBM Rational Doors

IBM Rational Doors es una herramienta de gestión de requisitos desarrollada por IBM que facilita la captura, el rastreo, el análisis y la gestión de cambios en la información.

Como características principales:

- Permite gestionar los cambios en los requisitos con un sistema de propuestas de cambios simple o con un flujo de trabajo personalizado.

[1] IBM Rational Doors - Página oficial: https://www.ibm.com/es-es/marketplace/requirements-management
[2] Visure Requirements - Página oficial: https://visuresolutions.es/herramienta-ingeniera-requisitos/
[3] Jama Product Development - Página oficial: https://www.jamasoftware.com/
[4] RequisitePro - Página oficial: https://www.ibm.com/developerworks/ssa/library/IBM_Rational_RequisitePro.html

- Permite enlazar requisitos con elementos de diseño, casos de prueba y otros requisitos para proporcionar una potente rastreabilidad.
- Puede integrarse con otras herramientas de Rational para proporcionar una solución más completa abarcando más funcionalidades, herramientas como IBM Rational Team Concert para automatizar flujos de trabajo y gestionar cambios o IBM Rational Quality Manager para la gestión de la calidad y la automatización de pruebas.

Doors trabaja con un modelo de datos *Requirements Modeling Framework*(RMF) [9] compuesto por módulos y conectados entre si por relaciones. En este modelo de datos los enlaces están orientados de abajo hacia arriba, esto ayuda al análisis de impacto y la trazabilidad, y también permite que se puedan agregar enlaces aunque el usuario actual no tenga acceso de escritura al módulo de nivel superior.

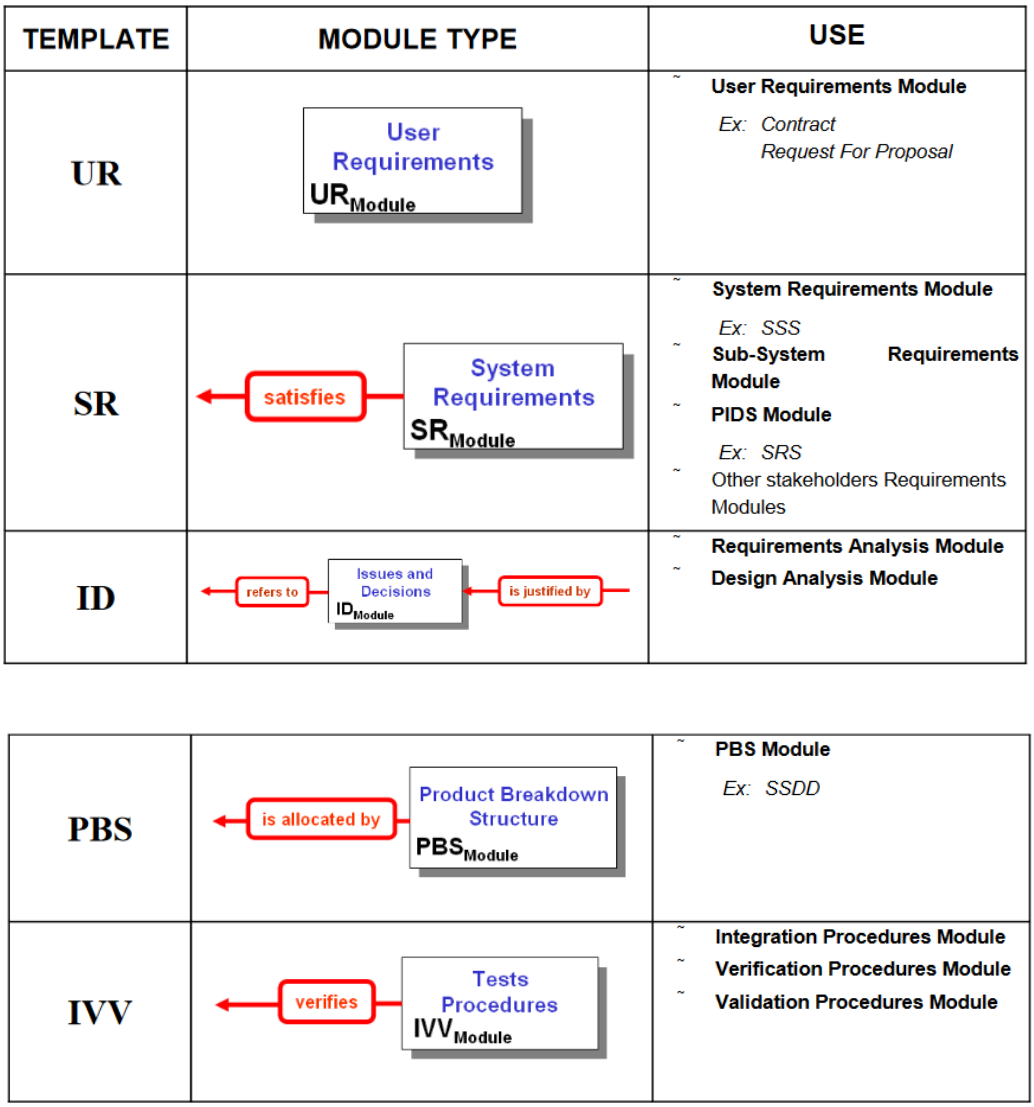


Figura 3.1: Lista de tipos de módulos genéricos de RMF

En la Figura 3.1 se puede observar la definición de las plantillas genéricas que ofrece RMF así como un ejemplo de uso para cada una ellas. Por ejemplo, la plantilla "IVV" implementa las defi-

niciones requeridas para procesar la descripción de las pruebas o la plantilla “SR” se utiliza para la definición de requisitos de sistema.

La plantilla define la naturaleza de la información que se almacena en el módulo y sirve para definir una estructura en los requisitos.

IBM Rational Doors permite identificar algunos requisitos como particularmente peligrosos para el proyecto y permite seguirlos más de cerca mediante una vista llamada “Analisis de riesgos”. Estos requisitos se pueden clasificar mediante el tipo de riesgo que conllevan (tecnológico, rendimiento, coste, plazo de entrega, etc.).

También permite gestionar la trazabilidad. En la Figura siguiente vemos un ejemplo en el cual relaciona 4 módulos: Requisitos del usuario >Requisitos funcionales >Plan de prueba funcional >Casos de prueba funcional.

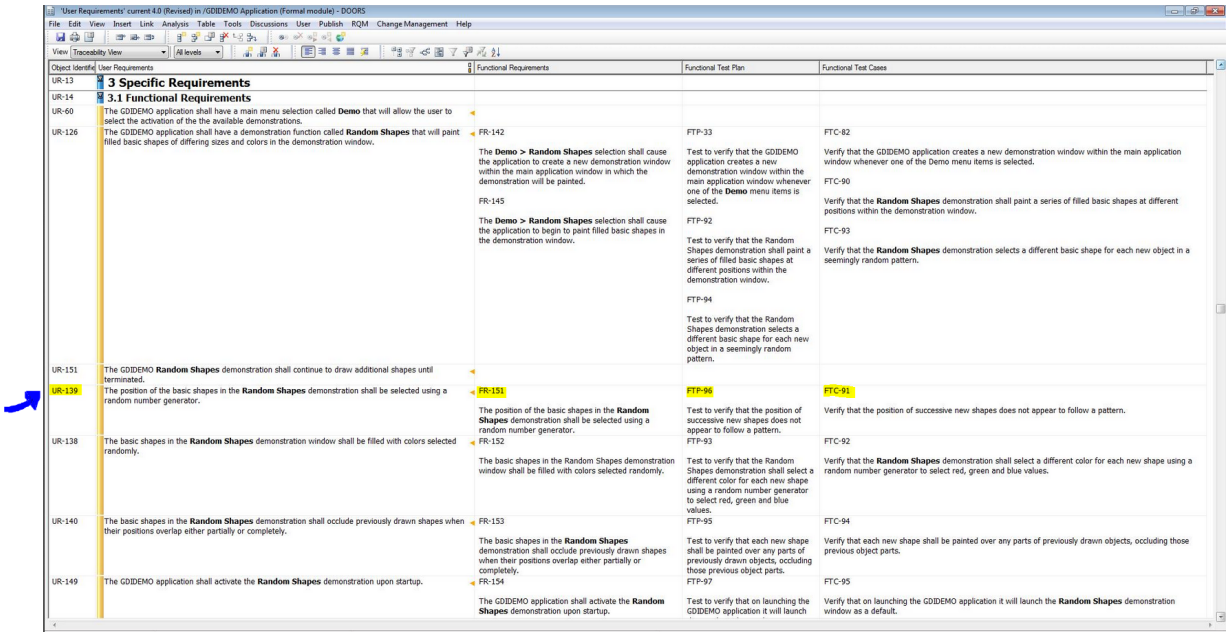


Figura 3.2: Trazabilidad desde IBM Rational Doors

En la imagen se ha resaltado el Requisito de Usuario UR-139 que especifica la siguiente funcionalidad: “La posición de las formas básicas en la demostración del ejemplo se seleccionará utilizando un generador de números aleatorios” y se puede observar que en la vista de trazabilidad de la herramienta aparece relacionado con:

- Requisito Funcional FR-151: “La posición de las formas básicas en la demostración del ejemplo se seleccionará utilizando un generador de números aleatorios”.

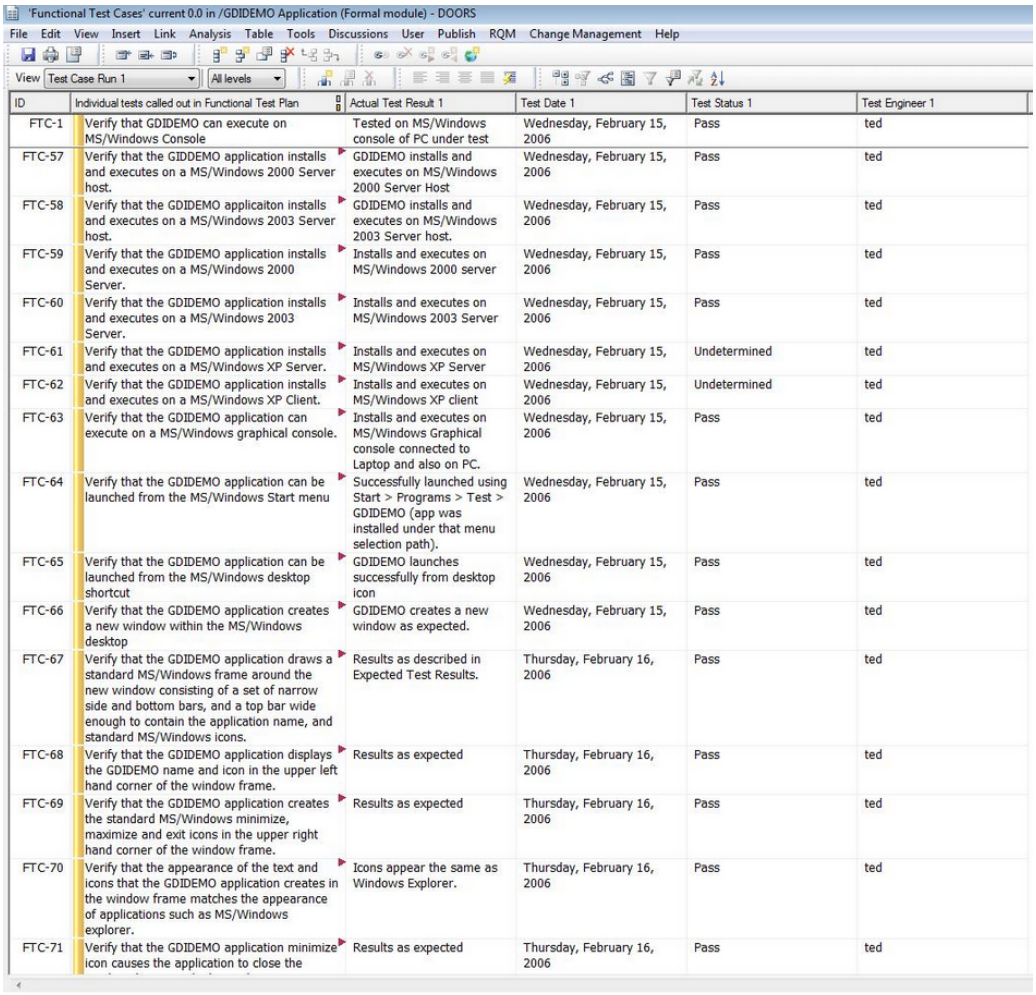
- Plan de Prueba Funcional FTP-96: “Prueba para verificar que la posición de nuevas formas sucesivas no parece seguir un patrón”.

- Caso de Prueba Funcional FTC-91: “Verificar que la posición de las nuevas formas sucesivas no parece seguir un patrón”.

Con este ejemplo se ilustra como se tratan en la herramienta Doors las relaciones entre los distintos tipos de requerimientos definidos en los diferentes módulos de RMF y como se puede visualizar su trazabilidad.

Doors permite relacionar los requisitos y la ejecución de los casos de prueba mediante una funcionalidad llamada Test Tracking Toolkit. Este módulo es capaz de almacenar y vincular fácilmente los requisitos a los resultados de las pruebas y si cambian los requisitos marca los enlaces como sospechosos para que los analistas revisen si tienen que actualizar el plan de prueba o los *testers* tienen que volver a probar dicha funcionalidad.

También permite crear Planes de Pruebas Funcionales donde agrupar diferentes pruebas funcionales y ejecutarlas como si fueran una batería de test.


'Functional Test Cases' current 0.0 in /GDIDEMO Application (Formal module) - DOORS

File Edit View Insert Link Analysis Table Tools Discussions User Publish RQM Change Management Help

View: Test Case Run 1 | All levels

ID	Individual tests called out in Functional Test Plan	Actual Test Result 1	Test Date 1	Test Status 1	Test Engineer 1
FTC-1	Verify that GDIDEMO can execute on MS/Windows Console	Tested on MS/Windows console of PC under test	Wednesday, February 15, 2006	Pass	ted
FTC-57	Verify that the GIDDEMO application installs and executes on a MS/Windows 2000 Server host.	GDIDEMO installs and executes on MS/Windows 2000 Server Host	Wednesday, February 15, 2006	Pass	ted
FTC-58	Verify that the GDIDEMO applicaiton installs and executes on a MS/Windows 2003 Server host.	GDIDEMO installs and executes on MS/Windows 2003 Server host.	Wednesday, February 15, 2006	Pass	ted
FTC-59	Verify that the GDIDEMO application installs and executes on a MS/Windows 2000 Server.	Installs and executes on MS/Windows 2000 server	Wednesday, February 15, 2006	Pass	ted
FTC-60	Verify that the GDIDEMO application installs and executes on a MS/Windows 2003 Server.	Installs and executes on MS/Windows 2003 Server	Wednesday, February 15, 2006	Pass	ted
FTC-61	Verify that the GDIDEMO application installs and executes on a MS/Windows XP Server.	Installs and executes on MS/Windows XP Server	Wednesday, February 15, 2006	Undetermined	ted
FTC-62	Verify that the GDIDEMO application installs and executes on a MS/Windows XP Client.	Installs and executes on MS/Windows XP client	Wednesday, February 15, 2006	Undetermined	ted
FTC-63	Verify that the GDIDEMO application can execute on a MS/Windows graphical console.	Installs and executes on MS/Windows Graphical console connected to Laptop and also on PC.	Wednesday, February 15, 2006	Pass	ted
FTC-64	Verify that the GDIDEMO application can be launched from the MS/Windows Start menu	Successfully launched using Start > Programs > Test > GDIDEMO (app was installed under that menu selection path).	Wednesday, February 15, 2006	Pass	ted
FTC-65	Verify that the GDIDEMO application can be launched from the MS/Windows desktop shortcut	GDIDEMO launches successfully from desktop icon	Wednesday, February 15, 2006	Pass	ted
FTC-66	Verify that the GDIDEMO application creates a new window within the MS/Windows desktop	GDIDEMO creates a new window as expected.	Wednesday, February 15, 2006	Pass	ted
FTC-67	Verify that the GDIDEMO application draws a standard MS/Windows frame around the new window consisting of a set of narrow side and bottom bars, and a top bar wide enough to contain the application name, and standard MS/Windows icons.	Results as described in Expected Test Results.	Thursday, February 16, 2006	Pass	ted
FTC-68	Verify that the GDIDEMO application displays the GDIDEMO name and icon in the upper left hand corner of the window frame.	Results as expected	Thursday, February 16, 2006	Pass	ted
FTC-69	Verify that the GDIDEMO application creates the standard MS/Windows minimize, maximize and exit icons in the upper right hand corner of the window frame.	Results as expected	Thursday, February 16, 2006	Pass	ted
FTC-70	Verify that the appearance of the text and icons that the GDIDEMO application creates in the window frame matches the appearance of applications such as MS/Windows explorer.	Icons appear the same as Windows Explorer.	Thursday, February 16, 2006	Pass	ted
FTC-71	Verify that the GDIDEMO application minimize icon causes the application to close the	Results as expected	Thursday, February 16, 2006	Pass	ted

Figura 3.3: Ejemplo de ejecución de una batería de tests desde la herramienta Doors

En la Figura 3.3 se puede observar un Plan de Pruebas Funcionales y el resultado de su ejecución. Tiene varias columnas donde se muestra la definición del caso de prueba, lo que verifica, la fecha de ejecución, el resultado y quien ha definido cada test.

En conclusión, Doors es una herramienta bastante completa que permite desde la especificación de requisitos hasta la ejecución de casos de prueba, pasando por la trazabilidad de éstos y el consecuente análisis de impacto de los cambios, pero no permite una visualización de la estructura global del producto a desarrollar así como poder contabilizar donde se acumulan los fallos, que nodos tienen más pruebas o han sufrido más cambios.

3.1.2. Visure Requirements

Visure Requirements es una solución de Ingeniería de Requisitos desarrollada por Visure Solutions para gestionar los procesos de requisitos dando soporte a la captura, análisis, especificación, validación y verificación, gestión y reutilización de éstos.

Como características principales:

- Permite la colaboración centralizada para realizar revisiones y aprobaciones por parte del cliente de forma automática mediante la firma electrónica.

- Permite importar y exportar datos a MS Word o MS Excel.

- Facilita la reutilización de requisitos permitiendo a los usuarios crear variantes de un requisito ya existente en la biblioteca.

- Tiene soporte para gestionar las solicitudes de cambio y permite el análisis de impacto.

- Proporciona trazabilidad entre los casos de prueba y los requisitos así como la definición de pruebas de aceptación o la captura automática de éstas desde MS Office.

Visure Requeriments permite organizar los requisitos en bloques. Un bloque es un conjunto de elementos del mismo tipo que comparten alguna característica, por ejemplo: requisitos funcionales, requisitos de sistema, pruebas de aceptación, etc. Un bloque es como un contenedor de requisitos y no es excluyente. Las relaciones entre requisitos se pueden definir entre los elementos pertenecientes a los bloques.

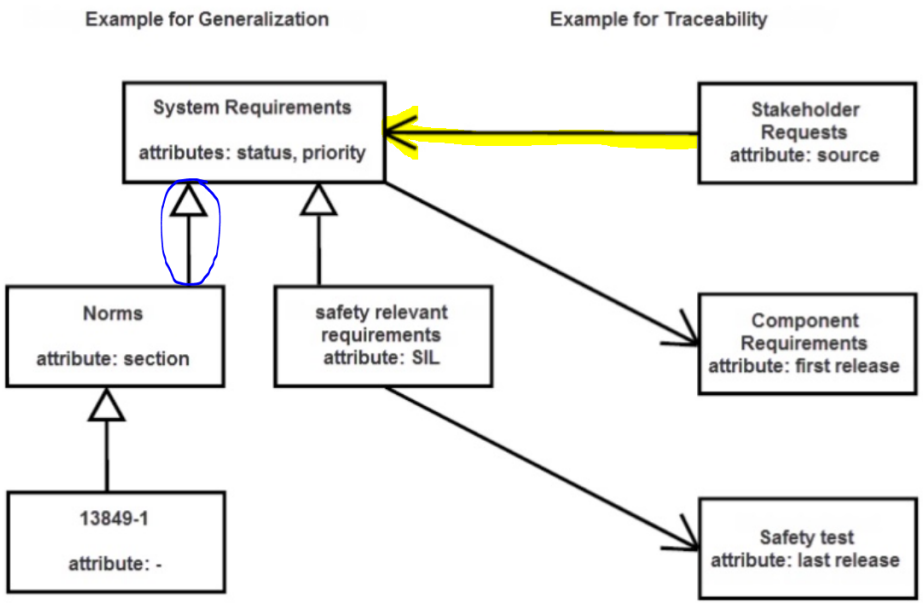


Figura 3.4: Ejemplo de relaciones en la herramienta Visure

En la Figura 3.4 se muestra un ejemplo de como se pueden relacionar los bloques dentro de la herramienta Visure Requirements. La flecha remarcada en azul muestra una relación de herencia, los atributos son heredados de los bloques padre. Al incluir un elemento en un bloque hijo automáticamente queda también incluido en el bloque padre. La flecha amarilla muestra una relación de trazabilidad.

Visure permite descubrir cuál fue el cambio exacto en un requisito que provocó que un enlace fuera sospechoso así como navegar a los elementos afectados por una solicitud de cambio o a los elementos relacionados con un riesgo.

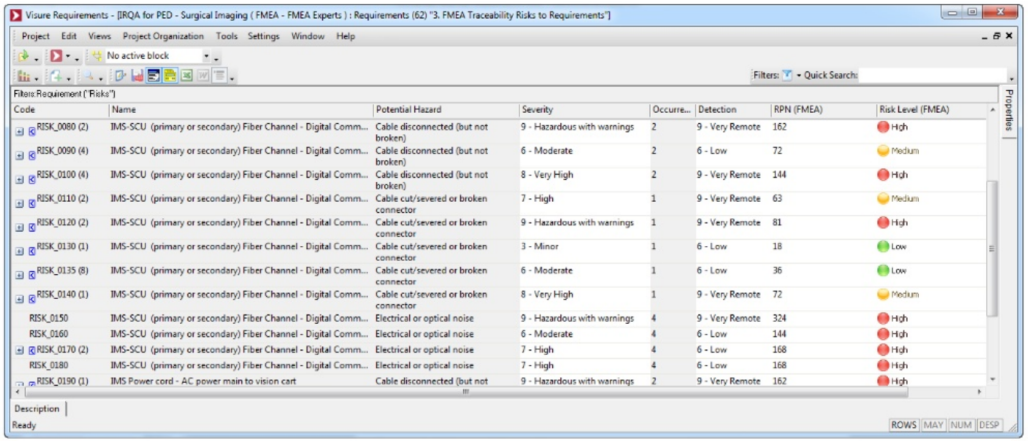


Figura 3.5: Ejemplo de trazabilidad del riesgo de los requisitos desde Visure

En la Figura 3.5 se muestra la trazabilidad del riesgo asociada a un canal de fibra. Cada fila es un riesgo identificado sobre ese requisito y se compone de un código, un nombre, el peligro potencial que implica, la severidad del riesgo, las veces que ha ocurrido, la posibilidad de detectarlo, RPN(FMEA) [10] que es una medida usada para evaluar el riesgo similar a la criticidad y por ultimo el nivel de riesgo que también depende del FMEA y según el valor obtenido clasifica los riesgos como débiles, medios o fuertes.

Identificar estos riesgos para posteriormente definir casos de prueba que verifiquen que no se producen es otra fase importante del análisis de requisitos de un producto.

En la Figura 3.6 aparecen los casos de pruebas definidos para comprobar y validar que el canal de fibra funciona correctamente.

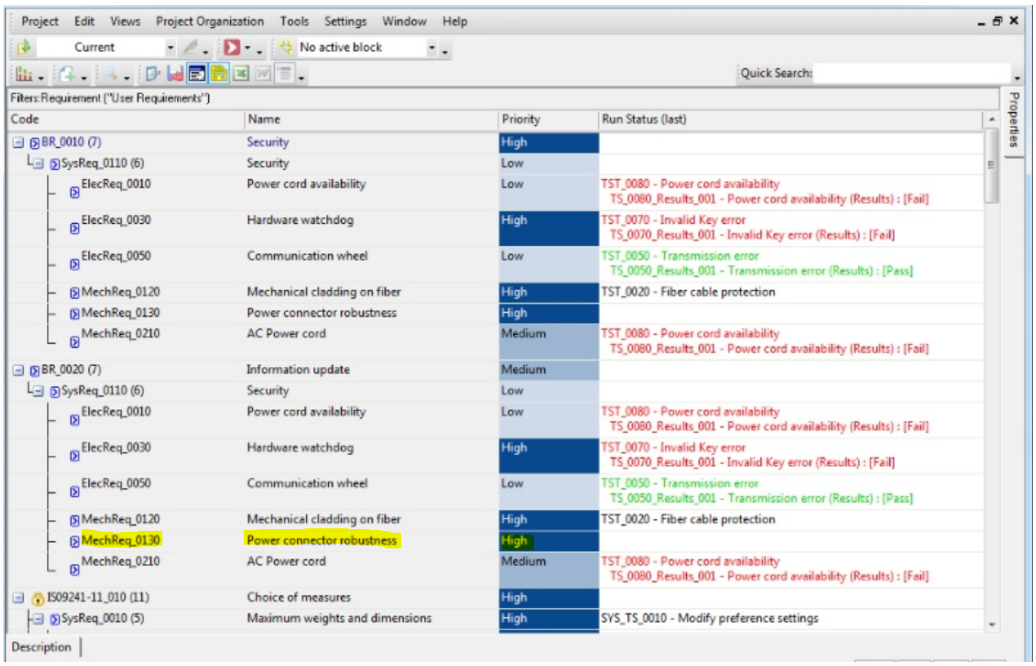


Figura 3.6: Ejemplo de ejecución de test en la herramienta Visure

Como se puede observar en la Figura 3.6 se han definido casos de prueba para comprobar que los posibles riesgos detectados anteriormente no se van a manifestar en el producto. Por ejemplo, se debería probar que la resistencia del conector de energía es correcta (MechReq_0130) para asegurar o al menos prevenir que el conector no se rompa (RISK_0130). La última columna de la derecha muestra el resultado de la ultima vez que se realizó el test así como el resultado obtenido.

La herramienta Visure Requirements tiene un tratamiento muy pobre para la gestión de cambios en los requisitos así como para la gestión y seguimiento de errores. Proporciona *add-ins* como Run Status para poder visualizar el estado de desarrollo en el cual se encuentran los requisitos identificando los estados mediante colores para que sea más sencillo de diferenciar o como Use Cases To Test que a partir de un caso de uso crea un escenario de test relacionado, pero a pesar de estos *add-ins* queda lejos de la herramienta interna SAPI en cuanto a gestión de la estructura del producto.

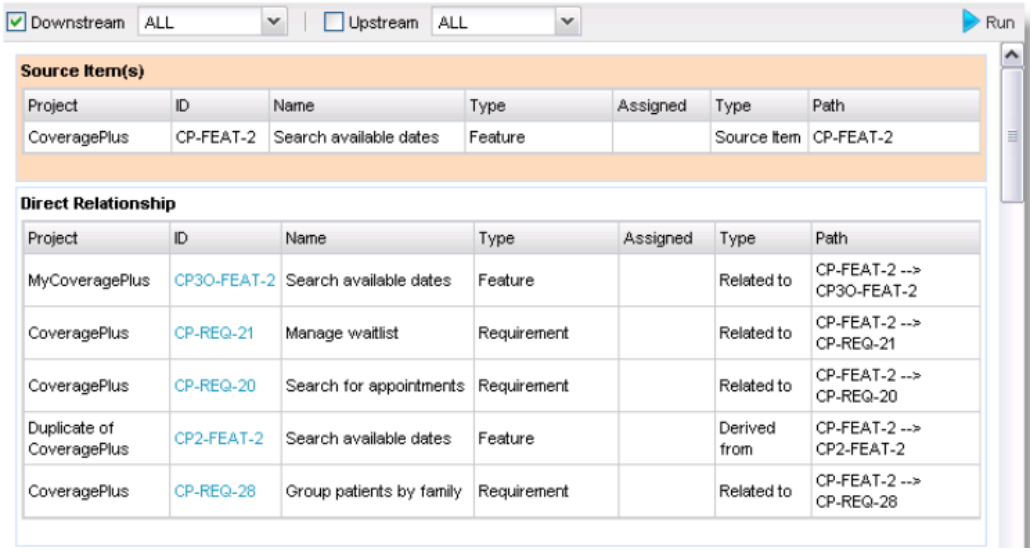
3.1.3. Jama Product Development

Jama Product Development es una solución desarrollada por Jama Software para guiar los aspectos estratégicos y operativos del desarrollo de productos software. Es compatible con todo el ciclo de vida del desarrollo, desde que surge la idea hasta que el producto se lanza al mercado.

Como características principales:

- Mide el rendimiento del equipo y optimiza las inversiones en productos y recursos adaptándose al ritmo de desarrollo.
- Permite estandarizar el proceso para garantizar que los equipos estén alineados y tengan lo que necesiten para garantizar el éxito del desarrollo.
- Sirve como herramienta de comunicación entre los distintos equipos para mejorar la eficiencia y la ejecución del proceso.
- Facilita el análisis de impacto para reducir los riesgos.
- Permite la trazabilidad de los requisitos.

Dependiendo de la organización que se use en Jama un elemento podría representar una característica, un requisito, un caso de uso, una tarea, un defecto o cualquier otro elemento que se necesite administrar. Las relaciones se utilizan para vincular elementos relacionados entre sí. Al crear relaciones entre los elementos, Jama permite evaluar fácilmente el impacto que un cambio en un elemento puede tener en otros elementos.


Downstream ALL | Upstream ALL | Run

Source Item(s)

Project	ID	Name	Type	Assigned	Type	Path
CoveragePlus	CP-FEAT-2	Search available dates	Feature		Source Item	CP-FEAT-2

Direct Relationship

Project	ID	Name	Type	Assigned	Type	Path
MyCoveragePlus	CP3O-FEAT-2	Search available dates	Feature		Related to	CP-FEAT-2 --> CP3O-FEAT-2
CoveragePlus	CP-REQ-21	Manage waitlist	Requirement		Related to	CP-FEAT-2 --> CP-REQ-21
CoveragePlus	CP-REQ-20	Search for appointments	Requirement		Related to	CP-FEAT-2 --> CP-REQ-20
Duplicate of CoveragePlus	CP2-FEAT-2	Search available dates	Feature		Derived from	CP-FEAT-2 --> CP2-FEAT-2
CoveragePlus	CP-REQ-28	Group patients by family	Requirement		Related to	CP-FEAT-2 --> CP-REQ-28

Figura 3.7: Analisis de impacto en la herramienta Jama

En la Figura 3.7 se observa el impacto que tendría un cambio en un requisito. Jama distingue dos tipos de impacto:

- Upstream ->En una relación se dice que el elemento que está *Upstream* podría afectar al elemento que estamos visualizando y como consecuencia en un informe de cobertura un cambio en este elemento marcaría como sospechosos todos los ítem que estuvieran por debajo de él, incluido el que visualizamos en la imagen.

- Downstream ->En una relación se dice que el elemento que está *Downstream* podría verse afectado por cambios en el ítem que estamos visualizando y como consecuencia en un informe de cobertura se marcarían como sospechosos todos los ítem que estuvieran por debajo si el ítem superior hubiera cambiado.

Jama permite ver el impacto de un cambio en un articulo seleccionado antes de realizar el cambio. Es una ventaja de implementar la trazabilidad del proyecto ya que puede ahorrar tiempo y garantizar que todo el equipo comprenda los posibles impactos. El análisis comienza desde el elemento seleccionado (resaltado con un fondo naranja) y sigue las relaciones de rastreo *Downstream* para mostrar los elementos que pueden verse afectados por el cambio.

Jama tiene una característica importante y es que permite configurar los flujos de trabajo para adaptarse a todo tipo de negocios, desde la gestión de requisitos hasta el seguimiento de defectos y la gestión de tareas del proyecto. Esto hace que se pueda configurar de varias maneras diferentes y permita gestionar mas allá del estado de un elemento. Esta es una característica deseable de la herramienta y que no hemos observado en los otros productos estudiados, salvo en SAPI.

Otra caracteristica que tiene es que permite crear vistas con múltiples niveles de trazabilidad.

Figura 3.8: Vista de cobertura en la herramienta Jama

En la Figura 3.8 se puede observar el requisito (REQ-100) que se divide en dos características (FEAT-100 y FEAT-101) y que a su vez cada una de ellas tiene asociados tres casos de prueba. Con estas vistas se puede visualizar la estructura del producto a nivel individual, es decir, para cada nodo del programa, pero no una visión global de la estructura completa.

En cuanto a cobertura de test permite definir Planes de Test en los que agrupar los casos de prueba. También tiene la opción de definir ciclos de pruebas que son una serie de ejecuciones de estos Planes de Test.

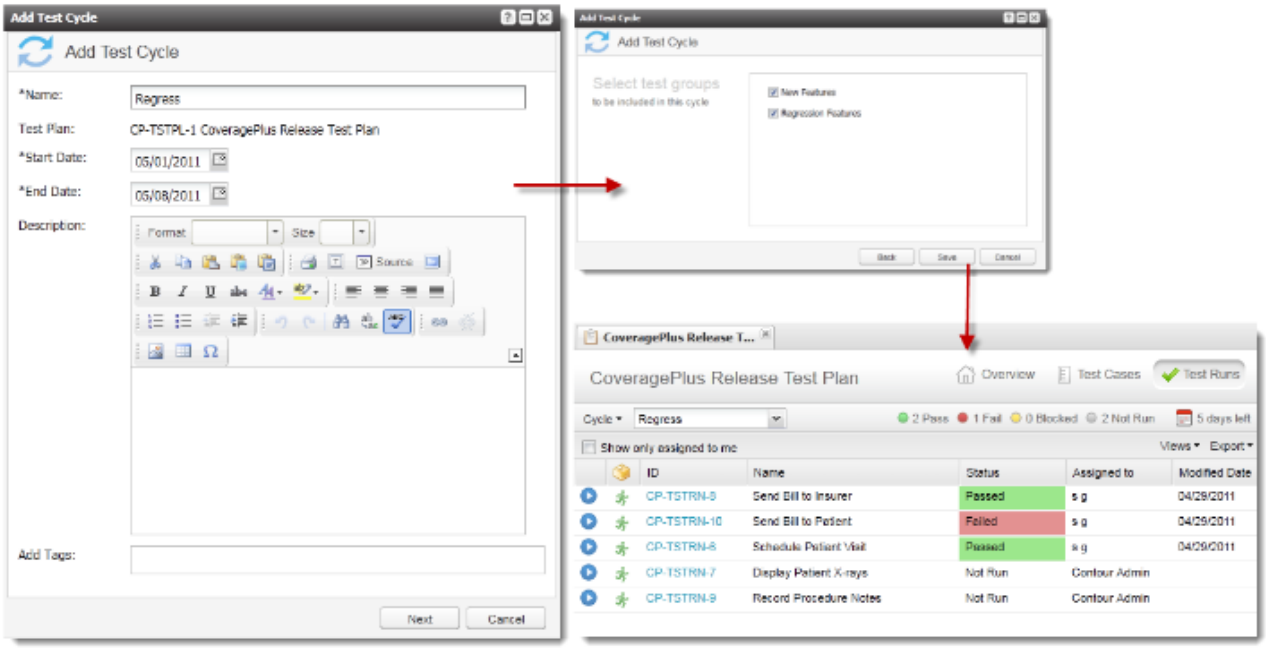


Figura 3.9: 1. Creando un ciclo 2. Añadiendo grupos de test al ciclo 3. Los resultados de ejecución

En la Figura 3.9 se muestra el flujo completo, desde la creación de los ciclos de test, pasando por la adición de pruebas hasta llegar al resultado de la ejecución. En la ventana de resultado se puede observar que permite exportar los resultados de los test a diferentes tipos de archivo. No tiene un *log* donde se almacenen los distintos resultados de ejecución para poder comparar, habría que hacerlo de forma manual mediante la comparación de la biblioteca de archivos guardados.

Jama es mucho más completa que el resto de las herramientas analizadas pero aun así, no tiene una gestión clara de la estructura del producto.

3.1.4. IBM RequisitePro

IBM Rational RequisitePro es una herramienta para trabajar con requerimientos tales como las especificaciones de casos de uso en formato de documentos de Microsoft Word. Proporciona un soporte de desarrollo y diseño integrado dirigido por el modelo con *Unified Modeling Language* (UML)[11].

Una vez descargado e instalado en el equipo, se accede a la ventana principal desde la cual se puede crear un documento de especificación de casos de uso. Este documento de especificación contiene las propiedades textuales del caso de uso , lo que incluye las propiedades siguientes: nombre, descripción breve, flujo de suceso básico, flujo de sucesos alternativo, condiciones previas, condiciones posteriores y requisitos especiales. Para ello previamente se deberían haber detectado los actores.

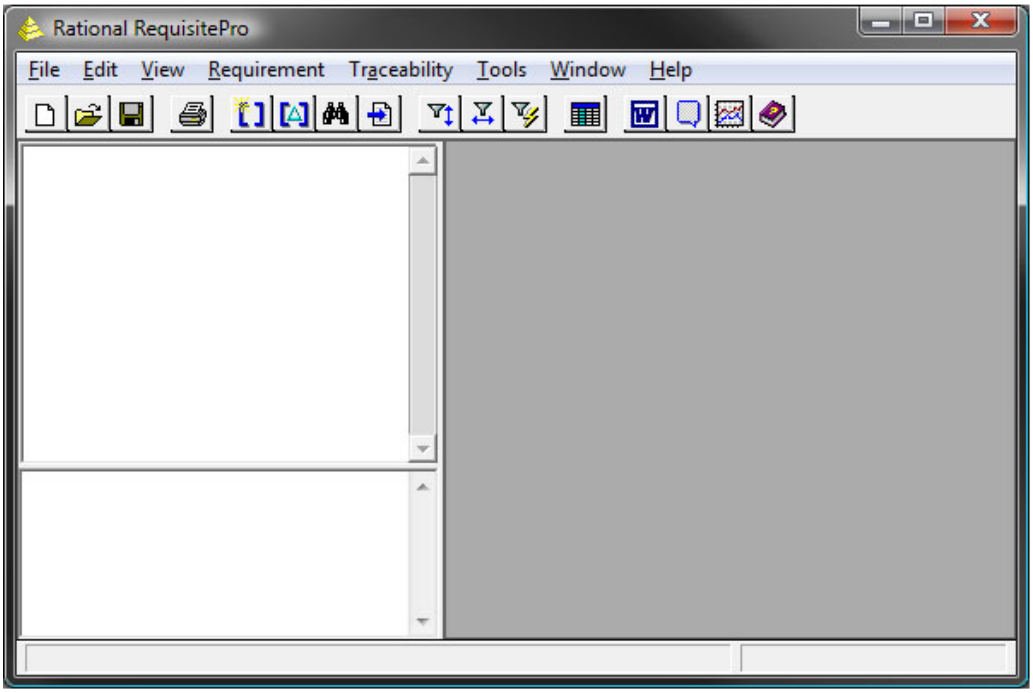


Figura 3.10: Ventana Principal de RequisitePro

En la Figura 3.10 vemos una captura de la ventana de inicio del programa. Es el punto de partida de la aplicación. Desde aquí se crean los documentos que posteriormente se visualizaran en Word, jerarquias de requerimientos para tenerlos organizados o matrices de trazabilidad para analizar el impacto del cambio.

Una vez están definidos los requisitos que el sistema debe satisfacer, vamos a ver como se organiza estructuralmente:

RequisitePro permite a los equipos organizar los requisitos de una manera funcional mediante la definición de diferentes tipos de requisitos. Por ejemplo, las reglas empresariales suelen incluir requisitos de alto nivel a partir de los cuales los equipos obtienen las necesidades del usuario, las características y los tipos de requisitos del producto. Los casos de uso dirigen los requisitos de diseño que se pueden utilizar para definir requisitos de software. Los requisitos de prueba se pueden obtener a partir de los requisitos de software y se dividen en procedimientos de prueba específicos. De esta manera, lo requisitos quedarían divididos en grupos más gestionables y significativos.

RequisitePro también permite crear jerarquías con relaciones padre-hijo que reflejan una agrupación lógica entre requisitos. Los requisitos padre son de nivel superior, más generales; los requisitos de nivel hijo son requisitos de nivel inferior, más específicos. Cada requisito hijo solo puede tener un padre, pero un requisito puede ser tanto un padre como un hijo, es decir, permite multinivel.

Otra de las características que contempla esta herramienta es la rastreabilidad. Ninguna expresión de un requisito es única, es necesario descomponer las necesidades del usuario en requisitos derivados que implican relaciones y por tanto, también es necesario poder gestionar esas dependencias. Por ejemplo, una característica del producto esta relacionada con uno o varios requisitos individuales para definir el comportamiento funcional especifico o un caso de prueba esta relacionado con uno o varios requisitos al cual verifican y validan. Es importante definir bien estas relaciones puesto que posteriormente garantizaran el éxito o el fracaso en cuanto a que puede afectar un cambio (análisis de impacto).

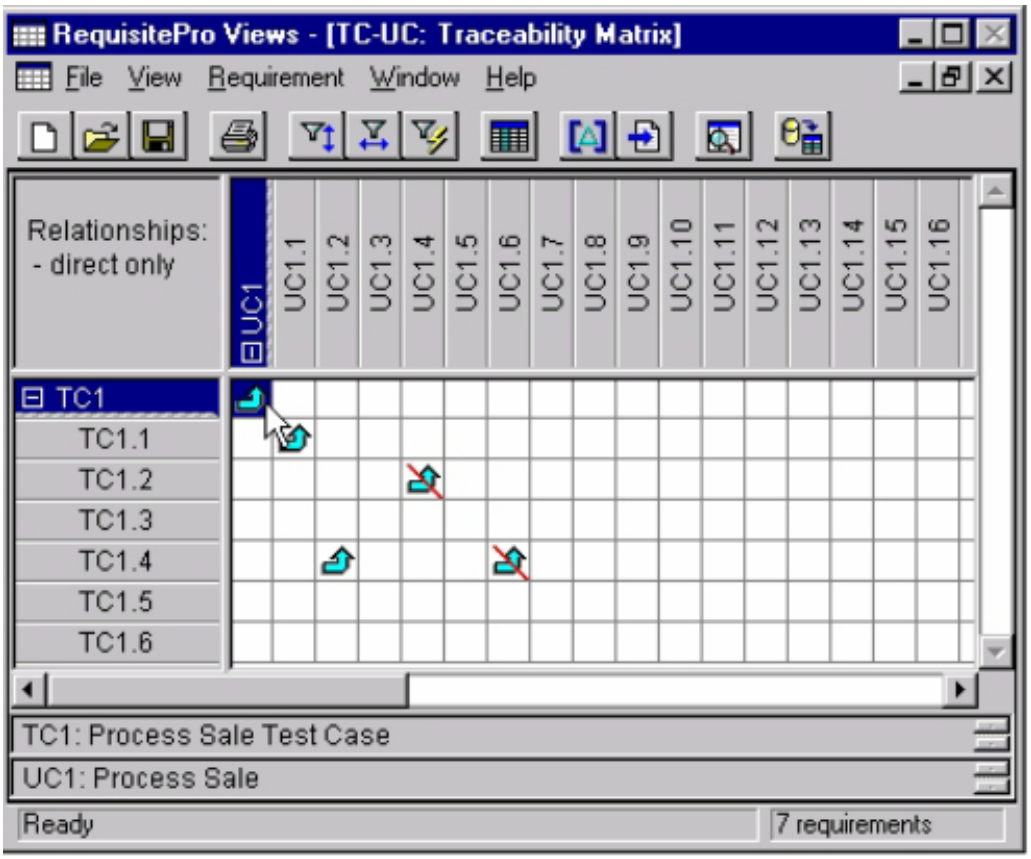


Figura 3.11: Matriz de trazabilidad entre casos de uso y casos de prueba en RequisitePro

En la Figura 3.11 vemos un ejemplo de la matriz de trazabilidad entre los casos de uso y los test de prueba de un determinado producto. En esta matriz se muestran las relaciones establecidas entre los elementos del eje X y los elementos del eje Y. Un enlace sospechoso (flecha roja) indica que el caso de prueba TC1.2 puede necesitar ser revisado debido a que ha habido un cambio en el caso de uso UC1.4. De esta manera los programadores podrían consultar los requisitos de las pruebas para garantizar que se siguen cumpliendo.

RequisitePro gestiona también el historial de requisitos lo cual permite hacer un seguimiento de qué, cuándo, por qué y quién de los cambios en los requisitos. Este es un paso importante para el análisis de impacto.

Por otra parte, esta herramienta no esta preparada para el trabajo en paralelo. La única manera de hacerlo es creando dos instancias separadas, trabajar de manera independiente y posteriormente unificar los cambios. Tiene atributos que pueden ayudar a gestionar desde dónde se ramificó un requerimiento y hacia dónde convergió el mismo para facilitar la comparación y el fusionado.

RequisitePro como tal, tampoco esta preparada para la gestión de fallos aunque se puede complementar con otra herramienta de IBM, ClearQuest [5], para la administración de cambios en el producto y el registro de defectos encontrados.

Rational ClearQuest puede gestionar todo tipo de actividad de cambio asociada con el desarrollo de software, incluidas mejoras, defectos y modificaciones de documentación. Al documentar y gestionar los problemas para su resolución, Rational ClearQuest proporciona una gestión de cambios de ciclo cerrado, ofrece un mejor control del proyecto y ayuda a mejorar la calidad del software. Rational ClearQuest gestiona desde la planificación de la prueba hasta la ejecución, pasando por la captura y el análisis del registro completo de los resultados de la prueba.

En la Figura 3.12 se muestra como se registra un error desde la plataforma web de ClearQuest.

[5]ClearQuest - Página oficial: https://www.ibm.com/developerworks/ssa/library/IBM_Rational_ClearQuest.html

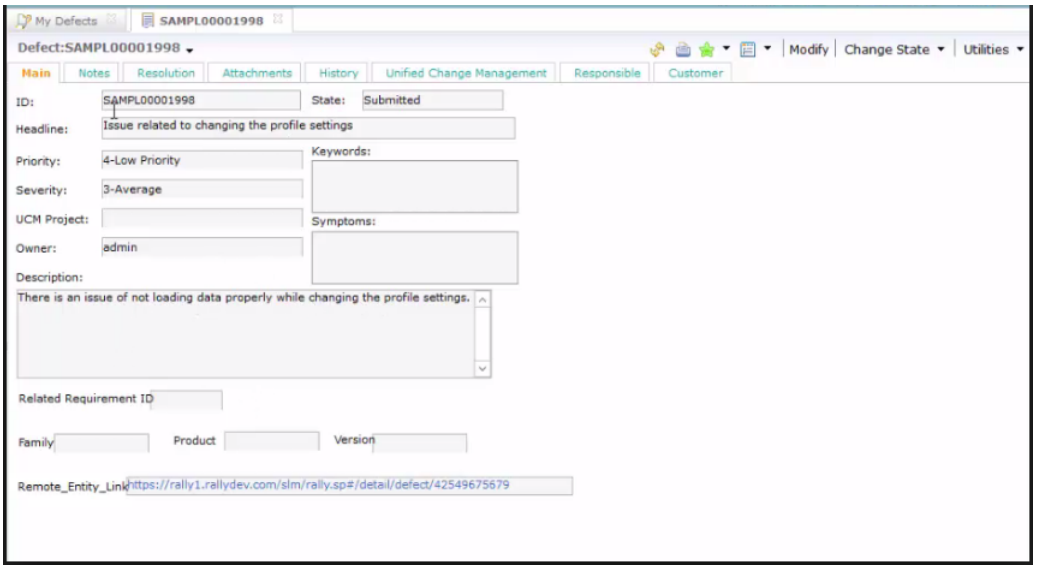


Figura 3.12: Registrar un defecto desde ClearQuest

Como se puede observar en la imagen, un defecto se puede relacionar con un requisito, se puede definir en que versión se detectó así como si el producto está compuesto por varios módulos a cual de ellos afecta.

En la definición del defecto también se le puede asociar una prioridad (como de importante es) así como una criticidad (medida para determinar como de grave es el error). Se le puede asociar a un responsable para que se encargue de su solución y tiene una pestaña donde se puede observar el historial de ese fallo.

Como complemento a la herramienta que se estaba estudiando es bastante completa y suple las carencias de ésta en cuanto a gestión de fallos e historial de cambios en los requisitos, pero sigue sin tener una definición de la estructura del producto, una vista donde se puedan observar sobre cada nodo del programa cuantas pruebas tiene, cuantas incidencias de fallo le han afectado, etc.

3.1.5. Conclusiones

Tras analizar las herramientas de gestión vistas en este capítulo, se concluye que la mayoría de ellas solo se centran en la gestión de requisitos sin prestar atención al desarrollo y mantenimiento del producto ni a la visualización de la evolución del producto.

No se ha encontrado ninguna herramienta de gestión que cumpla con los objetivos de este TFG y por ello el trabajo se ha centrado en buscar mecanismos de visualización y explotación de información, partiendo de la estructura de programa y datos disponible en la herramienta de la empresa.

CAPÍTULO 4
Mejora de los cálculos del GR

El punto de inicio de este trabajo ha sido la mejora de los cálculos que se mostraban en la vista estadística del Gestor de Requisitos del SAPI (ver Figura 2.2).

En un principio la idea era hacer las correcciones sobre la herramienta de gestión interna, SAPI, pero finalmente consideramos más oportuno montar una solución independiente y cuando todo funcionará trasladar los cálculos a la herramienta para no restringirnos de antemano a la tecnología del SAPI.

Para ello, antes de empezar a programar, se hizo un estudio previo del código contenido en la solución del SAPI y de la estructura de la base de datos SQL.

4.1 Solución SAPI

La solución de SAPI utiliza *Object-Relational Mapping (ORM)* [15] para hacer un mapeo automático de las tablas de la base de datos en entidades y así poder trabajar con ellas mediante programación orientada a objetos. Esta técnica facilita la labor al programador al poder acceder a la base de datos automáticamente y evita la escritura manual de consultas SQL y el tener que gestionar los datos que devuelve dicha consulta mediante *DataSet*, *DataTable*, etc.

Las ORM ofrecen una serie de ventajas al programador tales como rapidez en el desarrollo, abstracción de la base de datos utilizada y seguridad en la capa de acceso entre otras, pero también tienen dos grandes inconvenientes y son que el aprendizaje resulta complejo y que en entornos de gran carga penaliza el rendimiento debido a los procesos de transformación de las consultas.

La solución de estudio usa concretamente *Entity Framework (EF)* [12] y *.Net*. Para ello el primer paso es crear un modelo que asigna las entidades y relaciones que se definen en el modelo en las tablas de una base de datos.

En la Figura 4.1 vemos parte del modelo contenido en la solución de SAPI y que representa a la base de datos del caso de estudio. Las cajas son las tablas de la base de datos y las flechas representan las relaciones entre dichas tablas.

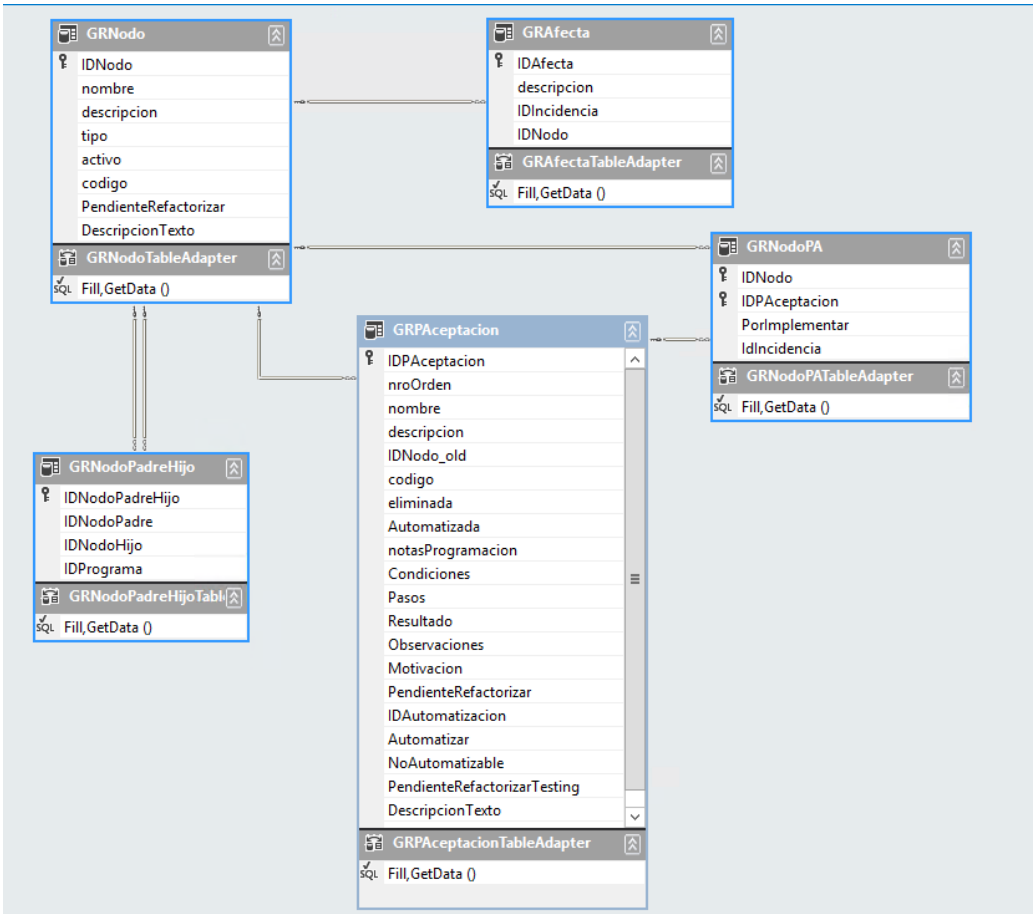


Figura 4.1: Modelo de la base de datos

En la Figura 4.1 se puede observar que la tabla de la base de datos “GRNodo”, la cual tiene una entrada para cada nodo definido en el Gestor de Requisitos, tiene también a su vez relaciones de distintas cardinalidades con el resto de tablas de la base de datos.

Una vez se tiene el modelo hay que definir la clase principal con la que se interactua, en este caso es una clase derivada de *DbContext* y que expone las propiedades de los distintos *DbSets*. De forma predeterminada este contexto administra las conexiones a la base de datos, abre y cierra estas según sea necesario.

Una vez se tiene el modelo y el contexto, ya se puede empezar a trabajar con EF para consultar, agregar o eliminar entidades a través de sus propiedades.

En la Figura 4.2 se muestra una consulta sacada del SAPI realizada mediante *Language Integrated Query (LinQ)* [13] sobre EF en la cual se obtiene el número de PA para cada nodo.

```
var NumPAsNodo = db.GRNodoPadreHijos.Where(n => n.IDPrograma == idPrograma).Select(n => n.GRNodo1).Distinct()
                 .Select(
                        p => new
                        {
                            Nodo = p,
                            numPAs = db.GRPAceptacion.Where(p2 => p2.GRNodoPAs.Where(n=> n.IDNodo == p.IDNodo).Any()  && (!p2.eliminada.HasValue || p2.eliminada.Value == false)).Count(),
                            numProp = db.GRCambioPA.Where(p2 => p2.GRPAceptacion.GRNodoPAs.Where(n => n.IDNodo == p.IDNodo).Any() && p2.Cerrada == null).Count(),
                            numPAsAutomatizadas = db.GRPAceptacion.Where(p2 => p2.GRNodoPAs.Where(n => n.IDNodo == p.IDNodo).Any()
                                && (p2.Automatizada.HasValue && p2.Automatizada.Value == true) && (!p2.eliminada.HasValue || p2.eliminada.Value == false)).Count(),
                            numPAsNoAuto = db.GRPAceptacion.Where(p2 => p2.GRNodoPAs.Where(n => n.IDNodo == p.IDNodo).Any()
                                && (p2.NoAutomatizable.HasValue && p2.NoAutomatizable.Value == true) && (!p2.eliminada.HasValue || p2.eliminada.Value == false)).Count(),
                            numInc = db.GRAfecta.Where(n => n.IDNodo == p.IDNodo).Select(n => n.IDIncidencia).Distinct().Count(),
                            numIDsFallo = db.GRAfecta.Where(n => n.IDNodo == p.IDNodo && n.Incidencias.IdTipoIncidencia == ID_TIPO_CORRECCION_FALLO).Select(n => n.IDIncidencia).Distinct().Count(),
                            numIDsMejoraNodo = db.GRAfecta.Where(n => n.IDNodo ==  p.IDNodo && n.Incidencias.IdTipoIncidencia == ID_TIPO_MEJORA).Select(n => n.IDIncidencia).Distinct().Count()
                        }).ToArray();
```

Figura 4.2: Consulta LinQ

Como es posible observar en la Figura 4.2 se deja de lado el lenguaje SQL y se utiliza C# para hacer las consultas a base de datos como si se tratara de objetos.

En este punto, sería interesante hacer hincapié en que EF facilita al programador abstraerse de la capa de persistencia y trabajar con la base de datos como si se tratara de objetos, todo ello mediante un lenguaje de programación propio como es *LinQ*. Pero también es importante resaltar que conlleva el trabajo previo de crear el modelo y el contexto así como de aprender a utilizar *LinQ* y tener en cuenta sus limitaciones.

Por ello, en la nueva aplicación se ha optado por no hacer uso de EF y usar SQL.

4.2 Solución TFGAnalisisDeDatos

Es la solución que se ha creado para obtener los cálculos correctos de los nodos del programa. Esta solución se ha implementado en C# desde *Visual Studio*[1].

4.2.1. Estructura de la solución

La nueva solución está estructurada en cinco clases: Conexion, AccesoBaseDatosSAPI, ComunConsultas, EntidadHojaExcelPadresHijos y el Program.

En la clase Conexion se define la ruta del servidor y de la base de datos para obtener la cadena de conexión, así como las operaciones básicas de abrir y cerrar conexiones utilizando la clase *SqlConnection* del API de *.Net*.

En la clase AccesoBaseDatosSAPI se han definido las distintas consultas SQL para obtener los datos de las tablas y las transformaciones necesarias entre el *Dataset* que devuelven y lo que se necesita en cada caso para la lógica de la aplicación.

En la clase ComunConsultas se han implementado los tratamientos a las diferentes excepciones que se pueden producir así como los errores que se van a mostrar en cada caso.

En la clase EntidadHojaExcelPadresHijos se crea una entidad de un objeto para almacenar información con la estructura concreta que posteriormente se va a utilizar.

Y la clase mas importante de la solución es el Program donde se define toda la lógica de la aplicación. Es donde se realizan todos los cálculos y las posteriores transformaciones de los datos a diferentes formatos tales como XML o JSON, para poderlos importar en las herramientas de visualización elegidas.

[1] Visual Studio - Página oficial: https://visualstudio.microsoft.com/

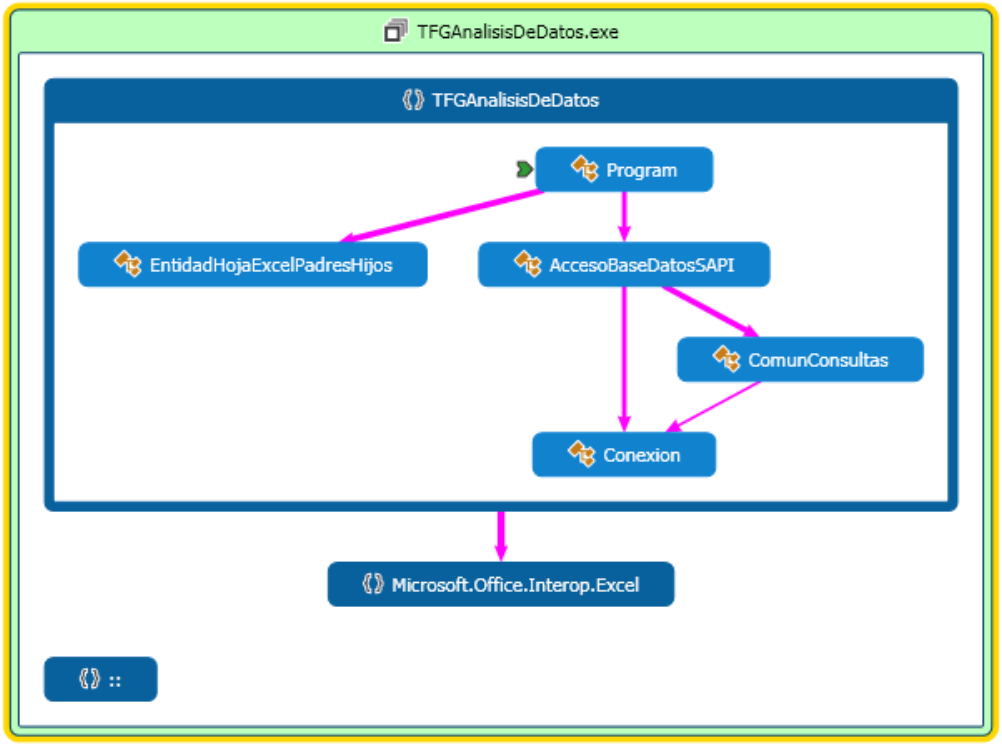


Figura 4.3: Esquema de la estructura interna de las clases que componen la solución.

En la Figura 4.3 se puede observar la estructura interna de las clases que componen la solución propuesta para corregir los cálculos, así como las dependencias entre las distintas entidades que vienen representadas mediante flechas de color rosa. Como se puede observar la clase Program es el punto de entrada al nuevo programa. Esta clase tiene dependencias directas con la clase AccesoBaseDatosSAPI a la cual se conecta mediante SQL para obtener los datos de los diferentes nodos y con la clase EntidadHojaExcelPadresHijos la cual utiliza para volcar la información calculada en una estructura propia y posteriormente poder persistirla.

4.2.2. Implementación de la lógica

Para la lógica de esta solución se ha utilizado la clase *Dictionary* [16] de C# como estructura de datos auxiliar porque representa una colección de claves y valores. En la clave se ha guardado el Id del nodo y en el valor una lista de Id que corresponden a sus nodos hijos.

Para rellenar los distintos clave-valor de este *Dictionary* se ha utilizado la técnica de la recursividad porque en la base de datos solo esta guardada la relación de un nodo con su padre inmediato, es decir, con el nodo inmediatamente superior a él.

También se han suprimido los ciclos del grafo saliendo del bucle en el caso de que el nodo recuperado ya estuviera dentro de la lista de nodos hijos.

Mediante estos pasos se ha obtenido un primer *Dictionary* en el cual ha quedado almacenada la estructura del caso de estudio y que servirá posteriormente para evitar cálculos repetidos en los valores de los distintos nodos.

Se va a utilizar un pequeño fragmento del caso de estudio que representa el nodo “N47836 - Ventana de configuracion de Plan de Atencion” y sus nodos hijos para ilustrar como funciona esta técnica:

```
<N47836 Nombre="Ventana de configuracion de Plan de Atencion">
  <N47838 Nombre="Pestaña Necesidades">
    <N00646 Nombre="Ventana de Seleccion de Profesionales" />
    <N00923 Nombre="Tablas maestras" />
  </N47838>
  <N47839 Nombre="Pestaña Objetivos">
    <N00923 Nombre="Tablas maestras" />
  </N47839>
  <N47840 Nombre="Pestaña Acciones">
    <N00923 Nombre="Tablas maestras" />
  </N47840>
  <N47841 Nombre="Pestaña Relaciones">
    <N49278 Nombre="Impresion de Relaciones" />
  </N47841>
  <N47843 Nombre="Caracteristicas comunes de las tablas " />
</N47836>
```

En la Figura 4.4 se puede observar la estructura de ejemplo transformada en un grafo dirigido.

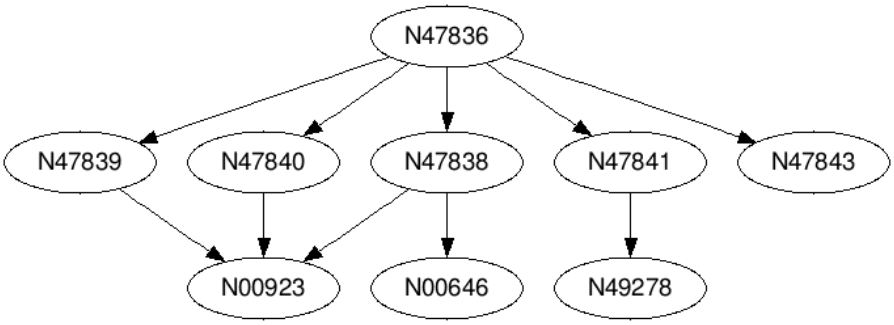


Figura 4.4: Grafo de la Ventana de Configuración de Plan de Atención.

Se puede comprobar que el nodo “N00923 - Tablas maestras” es a su vez hijo del nodo “N47839 - Pestaña Objetivos”, del nodo “N47840 - Pestaña Acciones” y del nodo “N47838 - Pestaña Necesidades”.

Al crear la entrada en el *Dictionary* para el nodo “N47836 - Ventana de configuracion de Plan de Atencion” en la clave se guarda el Id del nodo, es decir, “47836” y en el valor una lista de Id que corresponden a sus nodos hijos, en este caso concreto la lista de Id siguientes: “47839, 00923, 47840, 47838, 00646, 47841, 49278, 47843”. Al salir del bucle cuando un nodo ya se encuentra dentro de la lista se evita que se duplique (o en este caso, triplique) las entradas en la lista para el nodo N00923. De esta forma se evita información redundante que podría ocasionar que los cálculos posteriores no fueran correctos.

A continuación se ha procedido a crear un nuevo *Dictionary* para cada una de los valores que se debían de corregir en los cálculos de la pestaña “Automatización” del SAPI, es decir, uno para el número de PA por nodo, otro para el número de PS por nodo y así sucesivamente. En estos nuevos *Dictionary* la clave sigue siendo el Id del nodo y el valor es un entero que representa la cantidad de elementos de ese tipo que tiene en su propio nodo, sin tener en cuenta los que hereda de los nodos superiores en la jerarquía. De esta manera tenemos la información individualizada de cada nodo para posteriormente utilizarla en el calculo global.

En la Figura 4.5 se le ha añadido al grafo los valores en cuanto a número de PA. El valor es el número azul que aparece en la parte superior izquierda de cada nodo.

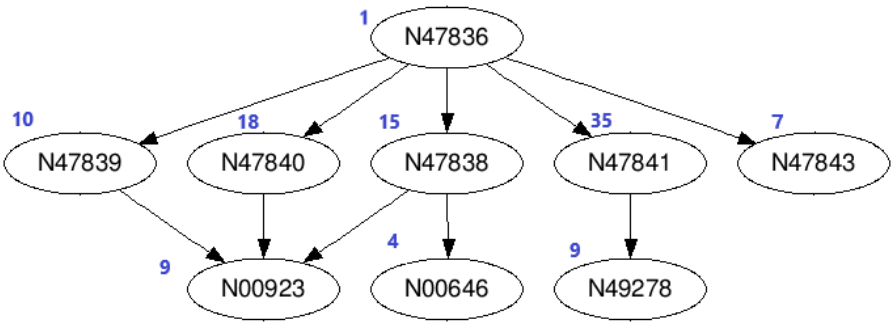


Figura 4.5: Grafo de la Ventana de Configuración de Plan de Atención con el valor para el número de PA.

En el caso de la Figura 4.5 se tiene una entrada para cada uno de los nodos en cada uno de los nuevos *Dictionary* donde la clave es el Id del nodo y el valor es el número de elementos propios de ese nodo. Por ejemplo, en el *Dictionary* donde se han almacenado el número de PA por nodo se tienen los siguientes valores:

IDNodo	NumeroPAs
47836	1
47838	15
646	4
923	9
47839	10
47840	18
47841	35
49278	9
47843	7

Por último solo queda recorrer todos los nodos guardados en el primer *Dictionary* como clave y sumar los valores para sus nodos hijos mas los suyos propios para tener el número total de elementos que le afectan.

En el fragmento del caso de estudio los valores para el número de PA de cada nodo teniendo en cuenta las que son propias y las que vienen heredadas de los nodos que quedan por debajo en la jerarquía se visualiza de la siguiente manera:

IDNodo	Nombre	NumeroPAs
47836	Ventana de configuración de Plan de Atención	108
47838	Pestaña Necesidades	28
646	Ventana de Selección de Profesionales	4
923	Tablas maestras	9
47839	Pestaña Objetivos	19
47840	Pestaña Acciones	27
47841	Pestaña Relaciones	44
49278	Impresión de Relaciones	9
47843	Características comunes de las tablas	7

Los valores en negro representan nodos finales, es decir, nodos que no tienen hijos. Los valores en rojo representan la suma de los valores para el nodo propio mas los valores de los nodos hijos.

4.2.3. Persistencia de los cálculos

En cuanto a la persistencia de los cálculos se ha creado un *DataTable* para luego exportarlo a un libro de Excel y tener todos los cálculos en una misma hoja.

```
/*Creo el dataTable resultado para exportarlo a Excel */
DataTable InformacionNodoToExcel = new DataTable("InformacionNodosResiPlus");
DataColumn numeroNodo = new DataColumn("numeroNodo", System.Type.GetType("System.Int32"));
DataColumn nombreNodo = new DataColumn("nombreNodo", System.Type.GetType("System.String"));
DataColumn numeroPAs = new DataColumn("numeroPAs", System.Type.GetType("System.Int32"));
DataColumn numeroPAsAutomatizadas = new DataColumn("numeroPAsAutomatizadas", System.Type.GetType("System.Int32"));
DataColumn numeroPSs = new DataColumn("numeroPSs", System.Type.GetType("System.Int32"));
DataColumn numeroPSsAutomatizadas = new DataColumn("numeroPSsAutomatizadas", System.Type.GetType("System.Int32"));
DataColumn numeroIncidencias = new DataColumn("numeroIncidencias", System.Type.GetType("System.Int32"));
DataColumn numeroIncidenciasFallo = new DataColumn("numeroIncidenciasFallo", System.Type.GetType("System.Int32"));
DataColumn numeroIncidenciasMejora = new DataColumn("numeroIncidenciasMejora", System.Type.GetType("System.Int32"));
DataColumn numeroIncidenciasNuevoRequisito = new DataColumn("numeroIncidenciasNuevoRequisito", System.Type.GetType("System.Int32"));
InformacionNodoToExcel.Columns.Add(numeroNodo);
InformacionNodoToExcel.Columns.Add(nombreNodo);
InformacionNodoToExcel.Columns.Add(numeroPAs);
InformacionNodoToExcel.Columns.Add(numeroPAsAutomatizadas);
InformacionNodoToExcel.Columns.Add(numeroPSs);
InformacionNodoToExcel.Columns.Add(numeroPSsAutomatizadas);
InformacionNodoToExcel.Columns.Add(numeroIncidencias);
InformacionNodoToExcel.Columns.Add(numeroIncidenciasFallo);
InformacionNodoToExcel.Columns.Add(numeroIncidenciasMejora);
InformacionNodoToExcel.Columns.Add(numeroIncidenciasNuevoRequisito);

/*Recorro la lista de nodos Resiplus y me guardo en un excel los datos referentes al numero de PAs y PSs por nodo teniendo en cuenta la herencia */
foreach (int nodo in listaNodosResiPlus)
{
    DataRow row = InformacionNodoToExcel.NewRow();
    row[numeroNodo] = nodo;
    consultarInformacionNodo(nodo);
    row[numeroPAs] = numeroPAsNodo;
    row[numeroPSs] = numeroPSsNodo;
    row[numeroPAsAutomatizadas] = numeroPAsAutomatizadasNodo;
    row[numeroPSsAutomatizadas] = numeroPSsAutomatizadasNodo;
    row[nombreNodo] = accesoBD.getNombreNodo(nodo);
    row[numeroIncidencias] = numeroIncidenciasNodo;
    row[numeroIncidenciasFallo] = numeroIncidenciasFalloNodo;
    row[numeroIncidenciasMejora] = numeroIncidenciasMejoraNodo;
    row[numeroIncidenciasNuevoRequisito] = numeroIncidenciasNuevoRequisitoNodo;
    InformacionNodoToExcel.Rows.Add(row);
}

guardarExcel(InformacionNodoToExcel);
```

Figura 4.6: Código C# para crear el DataTable

En la Figura 4.6, la cual es un fragmento de código C#, se puede observar que se crea un *DataTable* para almacenar los valores que son necesarios en las posteriores visualizaciones. Este *DataTable* esta compuesto por diez *DataColumn*, uno para el ID del nodo, otro para el nombre del nodo y uno para cada valor calculado. El número de filas se crea dinámicamente mediante un bucle *foreach* que recorre la lista de nodos del programa y crea una nueva fila para cada uno de ellos con sus correspondientes valores calculados.

Una vez ya se tiene el *DataTable*, se hace uso de la librería *Microsoft.Office.Interop.Excel*. Esta librería simplifica el acceso a objetos de la API de *Office* [2]. En la Figura 4.3 se podía observar la dependencia del proyecto con esta librería.

[2]Office - Página oficial: https://www.office.com/

```
public static void guardarExcel(DataTable dt)
{
    Excel.Application oXL = new Excel.Application();
    oXL.Visible = true;
    var misValues = System.Reflection.Missing.Value;
    Excel._Workbook oWB = oXL.Workbooks.Add(misValues);
    Excel._Worksheet wsh = oWB.Sheets.Add();
    wsh.Name = "InformacionPorNodo";
    wsh.Cells[1, "A"].Value2 = "IDNodo";
    wsh.Cells[1, "B"].Value2 = "Nombre";
    wsh.Cells[1, "C"].Value2 = "NumeroPAs";
    wsh.Cells[1, "D"].Value2 = "NumeroPAsAutomatizadas";
    wsh.Cells[1, "E"].Value2 = "NumeroPSs";
    wsh.Cells[1, "F"].Value2 = "NumeroPSsAutomatizadas";
    wsh.Cells[1, "G"].Value2 = "NumeroIncidencias";
    wsh.Cells[1, "H"].Value2 = "NumeroIncidenciasFallo";
    wsh.Cells[1, "I"].Value2 = "NumeroIncidenciasMejora";
    wsh.Cells[1, "J"].Value2 = "NumeroIncidenciasNuevoRequisito";
    for (int i = 0; i < dt.Rows.Count; i++)
    {
        wsh.Cells[i + 2, "A"].Value2 = dt.Rows[i][0];
        wsh.Cells[i + 2, "B"].Value2 = dt.Rows[i][1];
        wsh.Cells[i + 2, "C"].Value2 = dt.Rows[i][2];
        wsh.Cells[i + 2, "D"].Value2 = dt.Rows[i][3];
        wsh.Cells[i + 2, "E"].Value2 = dt.Rows[i][4];
        wsh.Cells[i + 2, "F"].Value2 = dt.Rows[i][5];
        wsh.Cells[i + 2, "G"].Value2 = dt.Rows[i][6];
        wsh.Cells[i + 2, "H"].Value2 = dt.Rows[i][7];
        wsh.Cells[i + 2, "I"].Value2 = dt.Rows[i][8];
        wsh.Cells[i + 2, "J"].Value2 = dt.Rows[i][9];
    }
    String pathFile = "E:\\Trabajo\\Silvia.Martos\\TFGAnalisisDeDatos\\TFGAnalisisDeDatosJunio2019";
    oWB.SaveAs(pathFile, Excel.XlFileFormat.xlWorkbookNormal);
    oWB.Close(true);
    oXL.Quit();
}
```

Figura 4.7: Código C# para persistir los cálculos

Como se puede ver en la Figura 4.7 mediante esta librería de interoperabilidad entre *Office* y *Visual Studio* es sencillo exportar los datos desde un *DataTable* a una hoja de Excel. Solo es necesario crear un objeto *Excel.Application* y recorrer las distintas filas del *DataTable* volcando sus valores a las correspondientes celdas del *Worksheet*. Posteriormente guardamos el archivo y ya lo tenemos preparado para importar en las distintas herramientas de visualización elegidas.

Se ha elegido una hoja de calculo para guardar la información por que *a priori* se había observado que todas las herramientas que se analizarán a continuación admiten este formato de entrada de datos y ademas es sencillo de trabajar con ella.

CAPÍTULO 5
Herramientas para la visualización de datos

En este apartado se introducen las distintas herramientas disponibles en el mercado y se analizan las soluciones que ofrecen para crear informes estadísticos sobre un producto. Todas ellas ayudan a representar grandes volúmenes de datos facilitando su correcta interpretación.

5.1 Selección de las herramientas

Con la gran cantidad de herramientas disponibles en el mercado se ha decidido comparar *Power BI, Qlik y Tableau* porque según los principales analistas tecnológicos son los lideres de soluciones de *Business Intelligence (BI)* [17].

Gartner Inc., la prestigiosa compañía consultora y de investigación en tecnologías de la información, los coloca en el cuadrante de *leaders* en su informe anual publicado en febrero de 2019 [18].

Figura 5.1: Cuadrante Mágico de Business Intelligence for Gartner

En este informe llevado a cabo por *Gartner Inc.*, las herramientas disponibles para BI se clasifican en cuatro cuadrantes. Cada uno de ellos representa una posición de los competidores dentro del mercado dependiendo de sus conocimientos para generar valor a sus clientes y a ellos mismos (eje X) y la habilidad que tienen para ejecutar con éxito su visión del mercado y la facilidad para adaptarse a los cambios (eje Y).

Las herramientas que se han elegido en este trabajo están situadas en el cuadrante derecho superior de la Figura 5.1

Este cuadrante conocido como "Lideres (*leaders*)" engloba herramientas que son maduras, completas y con una buena posición en el mercado del análisis empresarial.

Según el estudio los datos se obtienen mediante una encuesta que pasan a sus clientes de referencia y que aplicando una exhaustiva métrica valoran diferentes características de los proveedores tales como: facilidad de uso e interfaz atractiva, costes competitivos, visión global del producto y por supuesto la experiencia del consumidor.

En un principio se ha optado por estas herramientas de Escritorio pero tras los primeros informes se ha ampliado la búsqueda y se ha decidido incluir también una herramienta Web para ampliar el ámbito del trabajo.

A continuación se va a hacer una breve introducción a las herramientas mediante una descripción de cada una de ellas. En el Capítulo 7 se mostrará su aplicación a nuestro contexto de trabajo.

5.2 Power BI Desktop

PowerBI [1] es una solución de análisis empresarial que permite visualizar los datos y compartir la información con toda la organización, o insertarla en su aplicación o sitio web.

Esta desarrollada por Microsoft y permite la conexión a orígenes de datos locales o basados en la nube, como *Azure SQL DB, Excel, JSON, Dynamics 365* entre otros cientos.

Ofrece una licencia gratuita para estudiantes y una versión de prueba de 30 días y sino el precio por usuario y mes es de 8.40 €.

Obliga a una suscripción anual la cual asciende a 100.80 € por usuario y año.

Como principal ventaja observada frente al resto de herramientas comentar que se actualiza mensualmente incorporando nuevas funcionalidades y mejoras.

5.3 Tableau

Tableau [2] es una potente herramienta diseñada para los analistas y los usuarios avanzados que hacen preguntas profundas sobre los datos y descubren información .

Permite la conexión a datos en instalaciones físicas o en la nube. Tanto si se trata de *big data*, bases de datos SQL, hojas de cálculo o aplicaciones como *Google Analytics* y *Salesforce*.

[1] PowerBI - Página oficial: https://powerbi.microsoft.com/es-es/what-is-power-bi/
[2] Tableau - Página oficial: https://www.tableau.com/es-es

Ofrece una licencia gratuita para estudiantes y una versión de prueba de 14 días y sino el precio mensual por usuario es de aproximadamente 61.43 €. También obliga a la contratación de una licencia anual.

Destacar que permite cargar grandes volúmenes de datos en memoria y poder trabajar con ellos sin tener conexión.

5.4 QlikView/QlikSense

Tanto QlikView [3] como QlikSense [4] están desarrolladas por *QlikTech International* y forman parte de una amplia gama de productos diseñados para el análisis empresarial.

Permiten la conexión con servidores de datos *Apache, Azure SQL DB, Dropbox, REST y Presto* entre otros.

Ofrecen una licencia gratuita para uso personal y a nivel de empresa no pone los precios en su página web, te redireccionan a un formulario para que te pongas en contacto con su departamento de ventas. A través de la red se ha averiguado que su precio oscila alrededor de los 1.329 € lo cual da derecho a un *token* de uso ilimitado para un usuario. También se puede adquirir una licencia auto-gestionada mediante *SaaS (Software as Service)* [14] por aproximadamente 17,72 € por usuario y mes.

Destacar que una vez la instalas y tienes tus datos cargados tiene una opción que es "Generar Conocimientos" que mediante un motor cognitivo analiza los datos y ofrece vistas de estos mediante los gráficos que la herramienta considera más acertados para los datos de entrada que tiene.

5.5 JSFiddle.net

JSFiddle [5] comenzó como una pequeña aplicación de prueba en 2009 y actualmente esta consolidada como una plataforma para visualización de datos desde la web.

Es gratuita.

Destacar que tiene un *roadmap* publico desarrollado en Trello [6] en el cual puedes ver en que están trabajando los desarrolladores, que han realizado, lo que tienen pendiente... y lo más importante es que puedes votar que quieres que se implemente o sugerir nuevas características para que sean evaluadas por otros usuarios.

[3] QlikView - Página oficial: https://www.qlik.com/es-es/products/qlikview
[4] QlikSense - Página oficial: https://www.qlik.com/es-es/products/qlik-sense
[5] JSFiddle - Página oficial: https://jsfiddle.net/
[6] Trello - Página oficial: https://trello.com/

CAPÍTULO 6
Adaptación de los datos

En este apartado se exponen los datos de los que se dispone para analizar el producto de estudio y se someten a distintas transformaciones para poder ser utilizados con las herramientas descritas en el capitulo anterior.

Como se ha visto en el capitulo 4, los cálculos reales de cada nodo de la aplicación se hallan volcados en una hoja de Excel pero también es necesario tener las relaciones entre los nodos para poder visualizar el total del programa y poder navegar a través de su estructura.

6.1 Estructura de la aplicación

La aplicación de estudio tiene una estructura de grafo conexo, esto quiere decir que para dos nodos cualquiera de la aplicación (a,b) existe al menos un camino que une a con b.

También resaltar que tiene ciclos, por lo que el número de caminos posibles es infinito.

Teniendo en cuenta estas consideraciones en un primer momento se valoró que la mejor manera de representar esta estructura para poder trabajar con ella en las distintas herramientas de visualización de datos era mediante *eXtensible Markup Language (XML)*.

6.1.1. XML

Características de XML:

- XML esta concebido para el almacenamiento y distribución de información.

- Tiene la estructura lógica de un árbol de nodos, en el que los nodos son los elementos del documento.

- Tiene un nodo raíz que contiene al resto de nodos de la aplicación.

- Es un estándar internacionalmente conocido.

- Es fácilmente procesable.

- Su uso principal es para representar información estructurada de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa por muy diversos tipos de aplicaciones y dispositivos.

Por estas razones, y porque es un meta-lenguaje sencillo, se decidió trasladar la estructura de la aplicación estudiada a este formato.

Para ello dentro de la solución TFGAnalisisDeDatos se programaron unos métodos para crear el XML mediante la interfaz *Document Object Model (DOM)*.

En DOM un documento toma una estructura jerárquica que es similar a una estructura de árbol. El documento tiene un nodo raíz y el resto son ramas y hojas. Viene a ser una representación en memoria de un documento XML[19] .

En la Figura 6.1 se puede ver parte del documento XML que se generó para trasladar la estructura a las herramientas.

```
- <N47316 Nombre="Apartado Neuropsicologo ">
  - <N47318 Nombre="Funcionalidad comun de los apartados">
      <N48661 Nombre="Ventana Contenido de la Caja de Texto"/>
    - <N48694 Nombre="Subpestaña Plan de Atencion">
      - <N48695 Nombre="Tabla Acciones">
          <N47980 Nombre="Ventana de Seleccion de Personal por categorias"/>
          <N49212 Nombre="Ventana de seleccion de Accion"/>
        </N48695>
        <N48696 Nombre="Tabla Necesidades"/>
      - <N48697 Nombre="Tabla Objetivos">
          <N48906 Nombre="Historico del Objetivo. PAI de Historicos."/>
          <N49903 Nombre="Restricciones Tabla Objetivos en PAI de Periodos"/>
        - <N49907 Nombre="Evaluacion del Objetivo. PAI de periodos">
          - <N49905 Nombre="Formulario Individual de evaluacion">
              <N49908 Nombre="Registro de Acceso"/>
            </N49905>
          </N49907>
        </N48697>
      </N48694>
      <N48713 Nombre="Registro de Acceso"/>
    - <N48806 Nombre="Subpestaña Valoracion">
      - <N48657 Nombre="Boton Copiar">
        - <N48658 Nombre="arbol de seleccion de subapartados">
            <N00495 Nombre="arbol"/>
          </N48658>
        </N48657>
        <N49513 Nombre="Vista Filtrada por Escalas Vinculadas a Valoracio"/>
      </N48806>
    - <N48831 Nombre="Aviso de revision plan interdisciplinar">
        <N49055 Nombre="Boton Agenda"/>
      </N48831>
    - <N49362 Nombre="Corrector Ortografico">
        <N47623 Nombre="Corrector Ortografico de Office"/>
        <N49363 Nombre="Corrector Ortografico de Telerik"/>
      </N49362>
    </N47318>
</N47316>
```

Figura 6.1: Estructura en XML

El primer inconveniente que se encontró es que las herramientas elegidas tenían problemas de rendimiento para leer el archivo. Este archivo tenía más de 21.000 líneas y ocupaba alrededor de 1MB, lo cual no parece mucho, pero debido a su complejidad en cuanto a niveles del árbol era costoso de navegar por él y penalizaba mucho el tiempo de carga en las herramientas de visualización de datos.

PowerBi lo procesaba pero lo transformaba en un formato de tablas independiente en el cual no permitía navegar por la estructura que era el propósito perseguido.

Tableau tardaba alrededor de 15 minutos en procesar el archivo y luego la navegación era muy lenta.

Así que se optó por descartar XML y la segunda opción que se barajó fue *JavaScript Object Notation (JSON)* debido a que es un formato ligero de intercambio de datos [20].

6.1.2. JSON

Características de JSON:

- Es independiente de cualquier lenguaje de programación.

- Genera archivos de menor tamaño que XML.

- Es un formato más simple y en consecuencia más fácil de procesar.

En la Figura 6.2 vemos el mismo fragmento de la aplicación pero en formato JSON.


```
"N47316": {
  "-Nombre": "Apartado Neuropsicologo ",
  "N47318": {
    "-Nombre": "Funcionalidad comun de los apartados",
    "N48661": { "-Nombre": "Ventana Contenido de la Caja de Texto" },
    "N48694": {
      "-Nombre": "Subpestaña Plan de Atencion",
      "N48695": {
        "-Nombre": "Tabla Acciones",
        "N47980": { "-Nombre": "Ventana de Seleccion de Personal por categorias" },
        "N49212": { "-Nombre": "Ventana de seleccion de Accion" }
      },
      "N48696": { "-Nombre": "Tabla Necesidades" },
      "N48697": {
        "-Nombre": "Tabla Objetivos",
        "N48906": { "-Nombre": "Historico del Objetivo. PAI de Historicos." },
        "N49903": { "-Nombre": "Restricciones Tabla Objetivos en PAI de Periodos" },
        "N49907": {
          "-Nombre": "Evaluacion del Objetivo. PAI de periodos",
          "N49905": {
            "-Nombre": "Formulario Individual de evaluacion",
            "N49908": { "-Nombre": "Registro de Acceso" }
          }
        }
      }
    },
    "N48713": { "-Nombre": "Registro de Acceso" },
    "N48806": {
      "-Nombre": "Subpestaña Valoracion",
      "N48657": {
        "-Nombre": "Boton Copiar",
        "N48658": {
          "-Nombre": "arbol de seleccion de subapartados",
          "N00495": { "-Nombre": "arbol" }
        }
      },
      "N49513": { "-Nombre": "Vista Filtrada por Escalas Vinculadas a Valoracio" }
    },
    "N48831": {
      "-Nombre": "Aviso de revision plan interdisciplinar",
      "N49055": { "-Nombre": "Boton Agenda" }
    },
    "N49362": {
      "-Nombre": "Corrector Ortografico",
      "N47623": { "-Nombre": "Corrector Ortografico de Office" },
      "N49363": { "-Nombre": "Corrector Ortografico de Telerik" }
    }
  }
},
```


Figura 6.2: Estructura en JSON

Con este formato tampoco se fue capaz de trasladar la estructura de la aplicación a las herramientas de visualización.

Las herramientas cargaban la estructura pero no se lograba una navegación fluida, tardaba del rango de un minuto en mostrar una imagen nítida cada vez que se pulsaba en alguna parte de la pantalla, así que se siguió intentando mejorar esa carga.

6.1.3. Servidor propio de Base de Datos

Como tercera opción se decidió montar un servidor de base de datos SQL con las tablas que estaban involucradas en la estructura de la aplicación de estudio.

A grandes rasgos se creó un servidor local de base de datos mediante *SQL Server 2017* y posteriormente se importaron los esquemas y los datos de las tablas de la base de datos original que eran necesarias para trasladar la estructura a las herramientas de visualización.

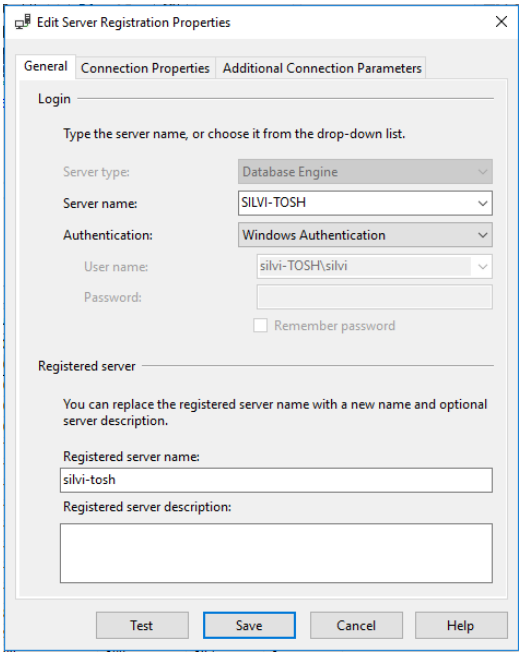


Figura 6.3: Servidor de Base de Datos

Con ello se tenía la estructura del producto en un servidor propio de base de datos y para acceder a esta estructura sólo era necesario iniciar una conexión al servidor de base de datos "pruebasTFG" desde las distintas herramientas.

De este modo, la conexión al servidor se establecía rápidamente pero a pesar de tener un editor avanzado para filtrar los datos, el formato esperado no eran consultas SQL, sino que requería de *Power Query* y el lenguaje *M*.

Como inconveniente más importante a este origen de datos, habría que resaltar que mediante el uso de *M* se creaban tablas intermedias con los datos ya filtrados que posteriormente serían el punto de partida del informe a generar, pero no se editaban en tiempo real, es decir, si se quería modificar los datos de entrada se debía de repetir la consulta en *M*, volver a crear las tablas intermedias con los nuevos datos y volver a generar el informe.

No es práctico para crear informes dinámicos. Tendría su utilidad en el caso de informes estáticos por lo que no se ha descartado totalmente este modo de carga de la estructura del programa puesto que puede resultar útil en algunos casos en que la información no sea excesivamente cambiante.

6.1.4. Tabla Plana en Excel

La cuarta opción y la que mejor resultados ha dado ha sido crear una tabla plana en una hoja de cálculo en la que se relaciona cada nodo con su nodo padre y el nivel de la jerarquía en el cual se encuentra.

A pesar de ser un archivo de mas de 6.600 filas, todas las herramientas de visualización lo cargan rápido, la navegación es fluida y cumple su función de permitir la navegación por la estructura de la aplicación de estudio.

En la Figura 6.4 se puede observar un fragmento del código C# con el cual se ha trasladado la estructura a una tabla aplanada.

```
public static void crearTablaPlanaResiPlus(int nodo)
{
    DataTable tablaPlana = new DataTable();
    tablaPlana.Columns.Add(new DataColumn("IDNodo"));
    tablaPlana.Columns.Add(new DataColumn("IDNodoPadre"));
    tablaPlana.Columns.Add(new DataColumn("Nivel"));
    int nivel = 0;
    DataRow newRow = tablaPlana.NewRow();
    newRow["IDNodo"] = nodo;
    newRow["IDNodoPadre"] = "0";
    newRow["Nivel"] = nivel;
    tablaPlana.Rows.Add(newRow);

    rellenarTablaPlana(nodo, tablaPlana, nivel);

    generateExcelFile(tablaPlana);
}

public static void rellenarTablaPlana(int nodo, DataTable tablaPlana, int nivel)
{
    DataSet resultado;
    int nodoRecuperado;
    nivel++;

    resultado = accesoBD.getNodosHijos(nodo);
    foreach (DataRow nodoHijo in resultado.Tables[0].Rows)
    {
        nodoRecuperado = Convert.ToInt32(nodoHijo["IDNodoHijo"]);
        DataRow newRow = tablaPlana.NewRow();
        newRow["IDNodo"] = nodoRecuperado;
        newRow["IDNodoPadre"] = nodo;
        newRow["Nivel"] = nivel;
        tablaPlana.Rows.Add(newRow);
        rellenarTablaPlana(nodoRecuperado, tablaPlana, nivel);
    }
}
```

Figura 6.4: Estructura aplanada

El método que crea la tabla plana recibe como parámetro el nodo inicial de la aplicación (padre absoluto) y para rellenar esa tabla llama al método "rellenarTablaPlana", donde recupera los nodos hijos del nodo recibido como parámetro de la llamada. Tal y como se puede observar en la Figura 6.4 se vuelve a utilizar la recursividad para navegar por los distintos nodos de la aplicación y obtener su nivel en la jerarquía.

De esta manera, ya se tienen adaptados los datos para generar las distintas visualizaciones que se van a mostrar en el capítulo siguiente.

CAPÍTULO 7
Visualización desde las distintas herramientas

En este capitulo se presentaran las gráficas mas conocidas y/o extendidas en la actualidad y se generaran visualizaciones de los datos de la aplicación de estudio a partir de ellas.

7.1 Tipos de gráfico

Quizá el más conocido es el gráfico de barras o diagrama de barras. En este, se presentan los datos en forma de barras contenidas entre los ejes cartesianos X e Y que indican los diferentes valores. El aspecto visual que indica el valor de los datos es la longitud de dichas barras, no siendo importante su grosor.

Otro gráfico muy habitual es el circular o por sectores. Se utiliza para representaciones porcentuales donde el círculo se divide en tantas partes como variables se tienen y siempre de tamaño proporcional a la frecuencia de ese valor dentro del total. Muestran la relación de las partes con el todo.

El gráfico de líneas se emplea para delimitar el valor de una variable dependiente respecto a otra independiente. Es usual que se emplee para observar la evolución temporal de una variable a través del tiempo.

Los gráficos de dispersión y de burbujas muestran relaciones entre dos o tres medidas cuantitativas. Los tamaños de las burbujas son útiles para resaltar rápidamente los valores.

Los gráficos de rectángulos pueden usarse para mostrar los datos jerárquicos como un conjunto de rectángulos anidados.

También para jerarquías se dispone del gráfico de proyección solar *(Sunburst)*. Su nombre se debe al parecido con el sol y los rayos, rayos desplegados de forma colorida y proporcional para una intuitiva interpretación.

Hay muchos tipos mas: gráficos de cascada, de embudo, de mapas coropleticos, etc.

Finalmente las visualizaciones se han generado con dos tipos de gráficos, burbujas y sunburst, se ha tomado la decisión de elegir estos gráficos porque se querían diferenciar los tamaños de los nodos en cuanto a cantidad de elementos que contienen de cada tipo estudiado y ademas poder navegar estructuralmente por la aplicación.

7.2 Gráfico de burbujas para visualizar las PA por Nodo

La primera visualización que se ha creado ha sido un gráfico de burbujas para analizar el número de pruebas de aceptación por nodo.

Para crear este gráfico se ha utilizado la herramienta Tableau y como archivos de entrada la hoja de Excel donde están los cálculos de cada nodo y una conexión al servidor de base de datos para la estructura del programa.

En la Figura 7.1 vemos representados todos los nodos de la aplicación.

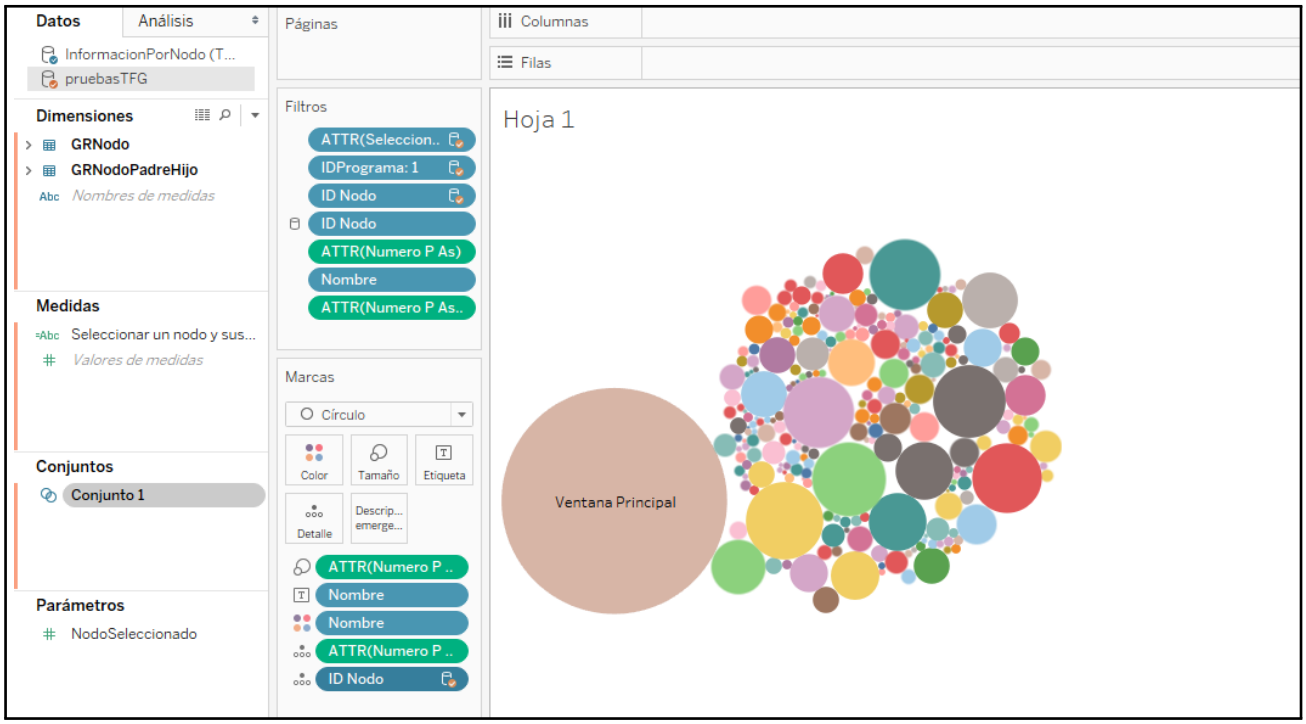


Figura 7.1: PA por nodo generado desde Tableau

Cada burbuja representa un nodo y el tamaño es proporcional al número de pruebas de aceptación que tiene ese nodo.

Como bien se puede observar en la Figura 7.1 se dispone de demasiada información para poderla interpretar de una forma cómoda. Es necesario poder aplicar filtros.

Para filtrar se ha creado un parámetro que se llama "nodo seleccionado" con el cual se apunta al nodo padre del que derivan todos los hijos que se van a mostrar en el informe. Solo los hijos directos, es decir, los nodos del siguiente nivel que descienden de ese nodo seleccionado. También se ha creado un campo calculado para recoger los ids de estos nodos.

En la Figura 7.2 se ven las pruebas de aceptación para los nodos de primer nivel de la aplicación de estudio.

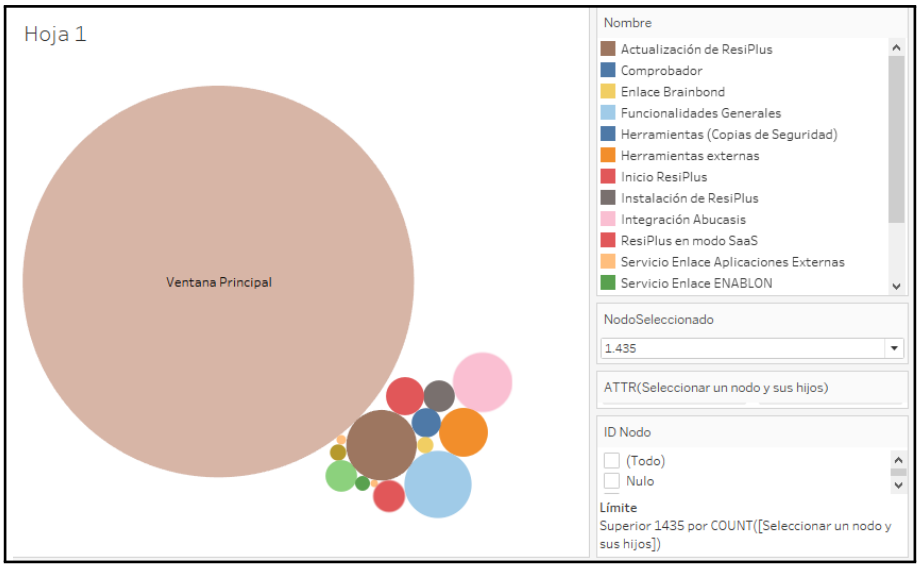


Figura 7.2: PA para nodos de primer nivel generado desde Tableau

Cada burbuja tiene una descripción emergente en la cual se puede ver el nombre del nodo, su Id, el numero de PA que contiene y el numero de esas PA que están automatizadas.

En este primer informe ya se puede observar donde esta el grueso de pruebas de aceptación del programa. El tamaño de la burbuja "Ventana principal" es mucho mas grande que la suma del tamaño del resto de burbujas que se están visualizando. Esto se debe a que el numero total de PA de la aplicación son 17.792 y solo este nodo ya contiene 16.308 (91,6 %).

Si se navega un nivel mas y se entra en el nodo "Ventana Principal" se puede observar como el total de las pruebas se van repartiendo hacia sus nodos hijos.

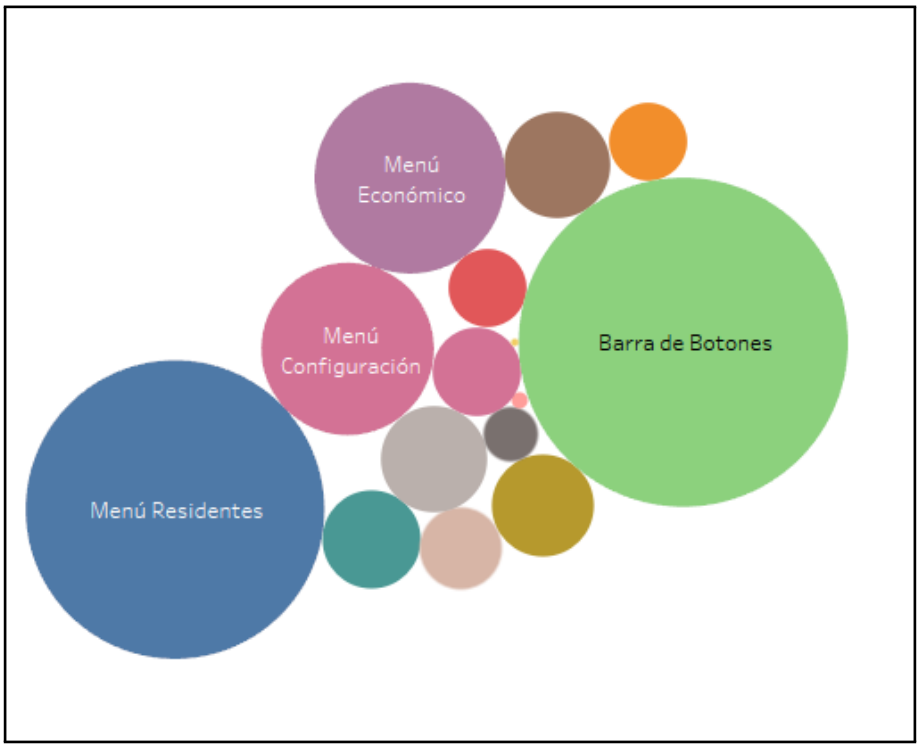
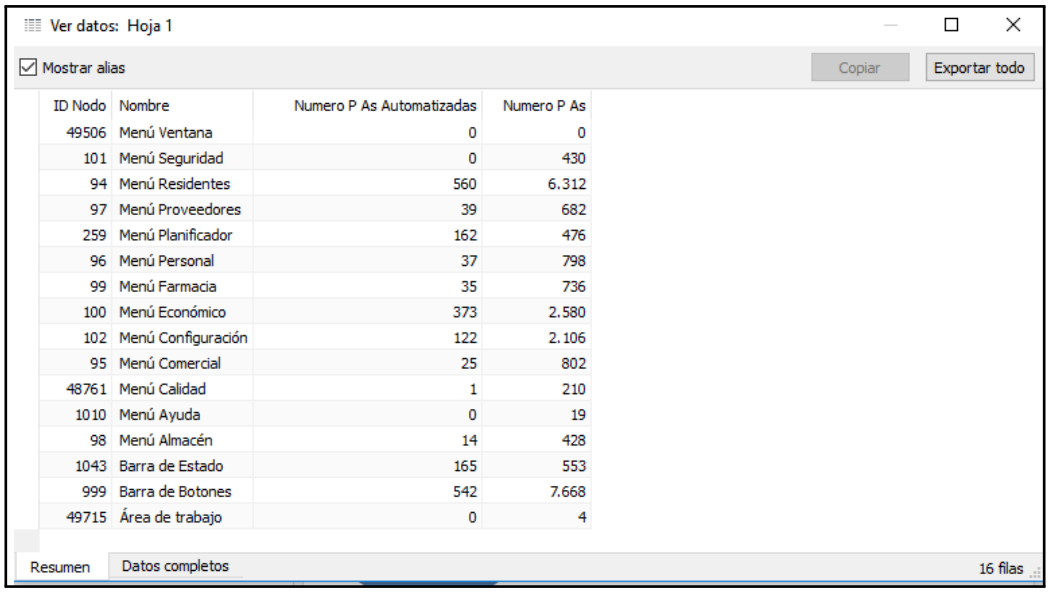


Figura 7.3: PAs para el nodo "Ventana Principal" generado desde Tableau



ID Nodo	Nombre	Numero P As Automatizadas	Numero P As
49506	Menú Ventana	0	0
101	Menú Seguridad	0	430
94	Menú Residentes	560	6.312
97	Menú Proveedores	39	682
259	Menú Planificador	162	476
96	Menú Personal	37	798
99	Menú Farmacia	35	736
100	Menú Económico	373	2.580
102	Menú Configuración	122	2.106
95	Menú Comercial	25	802
48761	Menú Calidad	1	210
1010	Menú Ayuda	0	19
98	Menú Almacén	14	428
1043	Barra de Estado	165	553
999	Barra de Botones	542	7.668
49715	Área de trabajo	0	4

Figura 7.4: Valores para el nodo "Ventana Principal" generado desde Tableau

De una manera visual se va observando que nodos tienen más cantidad de pruebas de aceptación y como estas se reparten en sus nodos hijos cuando este se expande. Como se puede observar en la Figura 7.4 también se puede obtener una vista de los datos para ese subconjunto de nodos y exportarlo a un dispositivo de salida.

Solo es necesario fijarse en el tamaño de la burbuja, mucho más claro e intuitivo que navegar entre un montón de valores en una hoja de cálculo o en un fichero de texto.

7.3 Gráfico Sunburst para visualizar las PA por Nodo

Resulta interesante generar un nuevo gráfico para la misma visualización de pruebas de aceptación por nodo, en este caso, con la herramienta web JSFiddle y partiendo de un archivo de texto con un formato concreto que requería la herramienta.

Para ello, era necesario introducir los nodos del programa como un *array* de datos donde cada nodo iba encerrado entre llaves y tenía los atributos *id, parent, name y value*. De nuevo ha habido que transformar los datos y ha sido necesario crear un *.txt* para poderlo trasladar a la aplicación web.

Tras solucionar distintos problemas a través del inspector del navegador, dado que no cargaba el gráfico y tampoco daba ningún mensaje, se ha conseguido crear la gráfica *sunburst*.

En la Figura 7.5 se observan todos los nodos de la aplicación.

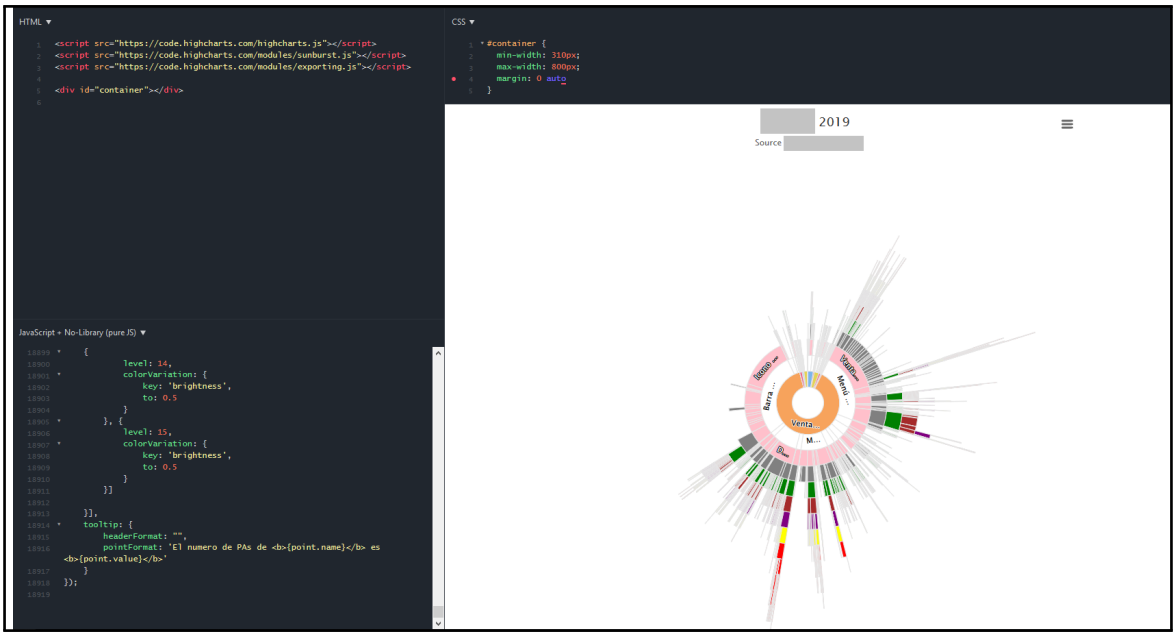


Figura 7.5: PAs por nodo generado desde JSFiddle

Si se hace *click* en cualquiera de los cuadrados de los anillos, este se expande y pasa a ser el centro del gráfico, mostrando solo los nodos que quedan por debajo de él en la jerarquía. El símil para explicar el funcionamiento sería como si el anillo se rompiera en sus piezas contribuyentes.

En la Figura 7.6 se ha pulsado en "Ventana Principal" y la gráfica ha quedado tal y como se observa a continuación:

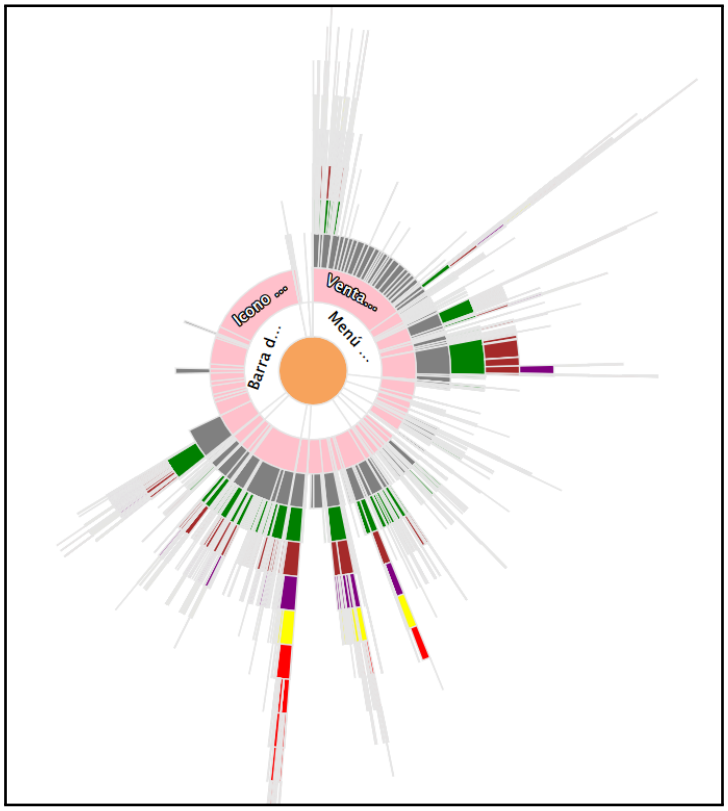


Figura 7.6: PAs para el nodo "Ventana Principal" generado desde JSFiddle

Cada uno de los anillos concéntricos representa un nivel de jerarquía en la aplicación. También se puede ver como los anillos externos se relacionan con los anillos internos.

Si posicionamos el ratón encima de uno de los cuadrados que forman anillos aparece un *tooltip* explicativo con el numero de PA y el nombre del nodo.

En la Figura 7.6 el anillo central naranja corresponde ahora al nodo "Ventana Principal" y todos los anillos restantes son hijos de este nodo. El primer anillo concéntrico (anillo blanco) representa a los nodos hijos de primer nivel. El segundo anillo concéntrico (anillo rosa) representa a los nodos hijos de segundo nivel. Y así sucesivamente hasta llegar a los nodos finales de la aplicación.

De igual manera que en el gráfico de burbujas creado con Tableau se puede observar que es posible navegar por los distintos niveles de la aplicación visualizando donde se concentran el mayor numero de pruebas de aceptación.

La conclusión alcanzada es, que ambos gráficos, el de proyección solar y el de burbujas, son buenas visualizaciones para representar datos jerárquicos y resumir la información en áreas proporcionales a los valores correspondientes (en estas gráficas ha sido numero de PAs pero también se podría utilizar con número de PSs o número de incidencias).

Por ejemplo, estas visualizaciones le serían útiles a un analista ya que podría ver claramente cuantas pruebas de aceptación tiene una funcionalidad del programa y desglosar navegando por los distintos niveles exactamente que elementos abarcan más o menos requisitos.

También se podrían emplear para la toma de decisiones de un gerente que tiene que decidir que requisito se debe implementar y quiere saber que artefactos se van a ver afectados y de que cantidad de pruebas dispone para comprobar que nada ha dejado de funcionar según lo esperado.

7.4 Gráfico de burbujas para visualizar las Incidencias por Nodo

El siguiente gráfico que se va a incluir en el estudio está construido desde PowerBi y toma como datos de entrada la hoja de cálculo con los valores de los nodos y la tabla plana para cargar la estructura de la aplicación.

En la aplicación de estudio las incidencias pueden ser de fallo, de mejora, de nuevo requisito u otro tipo de incidencia. En esta visualización se va a mostrar el total de incidencias por nodo, es decir, la suma de los distintos tipos de incidencia enumerados anteriormente.

También se ha utilizado el gráfico de burbujas, puesto que resulta interesante para poder comparar a posterior las ventajas e inconvenientes de las distintas herramientas con respecto a los mismos gráficos.

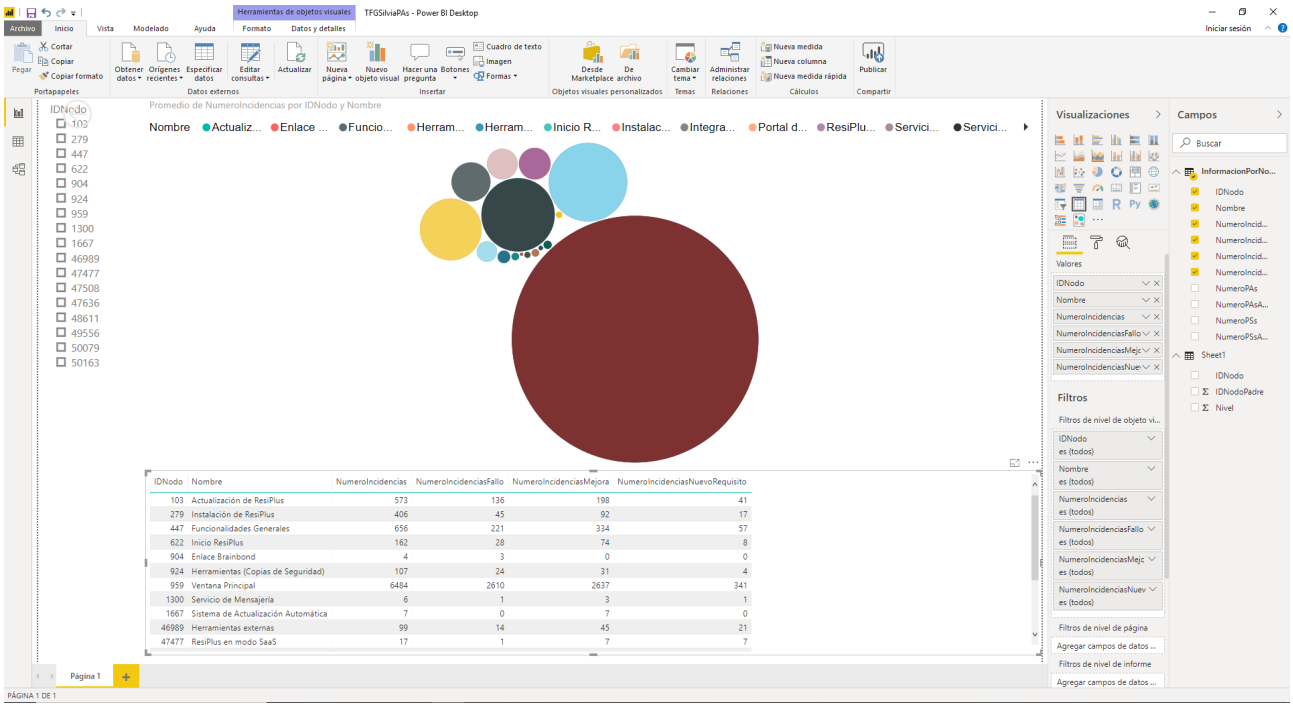


Figura 7.7: Incidencias por nodo generado desde PowerBi

Como se puede observar en la Figura 7.7 se han creado tres vistas en el informe inicial.

1. La primera de la izquierda es el identificador del nodo a seleccionar. Esta implementada con un *HierarchySlicer* y permite filtrar los nodos diciéndole de quien son hijos.

Hierarchy Slicer es una de las muchas aplicaciones compatibles con PowerBI desarrolladas por usuarios externos para ampliar una funcionalidad y compartida con el resto de usuarios a través de la misma herramienta.

Todas estas aplicaciones estan documentadas y son accesibles para los usuarios de PowerBi Pro.

Esta en concreto, *Hierarchy Slicer*, sirve para crear una jerarquía de campos diferentes y utilizarla como segmentación de datos para filtrar otros elementos del informe. Como dice su creador, Jan Pieter Posthuma, "cada nivel se puede expandir o contraer para una navegación optima y para buscar y seleccionar el atributo correcto de la segmentación de datos" [21].

En esta visualización queremos que se muestre solo los hijos de primer nivel, es decir, los descendientes directos del nodo que le hemos pasado como filtro.

2. La vista inferior muestra en formato tabla los datos de los nodos que se están visualizando.

3. Y, por último, la vista gráfica de los datos muestra, en burbujas de tamaño proporcional, el número de incidencias de cada uno de los nodos que se están analizando.

Podemos situar el cursor encima de cualquier burbuja y se muestra el nombre del nodo y el número de incidencias que tiene.

Para hacer explosionar un nodo y visualizar la información que contiene habría que realizar la siguiente secuencia de pasos:

a. Seleccionar la vista de la izquierda.

b. En el filtro seleccionar el nuevo nodo Padre (nodo a explosionar).

c. Para que se actualice la vista gráfica es necesario pulsar el icono de la esquina superior derecha.

En la Figura 7.8 se ha hecho explosionar el nodo "Ventana Principal" siguiendo los pasos definidos mas arriba.

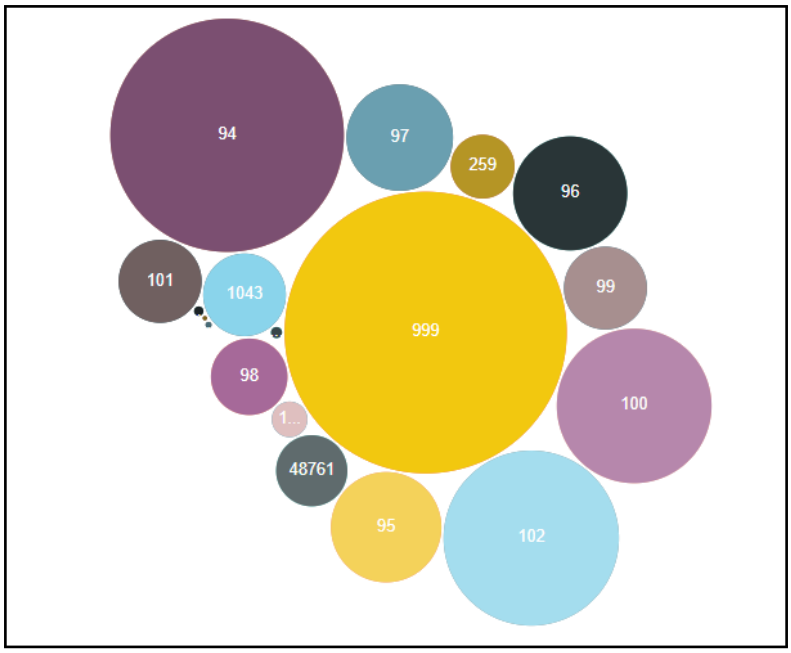


Figura 7.8: Incidencias para el nodo "Ventana principal" generado desde PowerBi

IDNodo	Nombre	NumeroIncidencias	NumeroIncidenciasFallo	NumeroIncidenciasMejora	NumeroIncidenciasNuevoRequisito
94	Menú Residentes	3186	1245	1335	204
95	Menú Comercial	712	302	270	37
96	Menú Personal	760	269	337	45
97	Menú Proveedores	662	268	269	36
98	Menú Almacén	342	167	145	20
99	Menú Farmacia	402	164	169	44
100	Menú Económico	1398	537	565	84
101	Menú Seguridad	402	147	176	38
102	Menú Configuración	1795	571	718	152
259	Menú Planificador	238	112	92	17
999	Barra de Botones	4650	1835	1960	248
1010	Menú Ayuda	74	8	25	3
1043	Barra de Estado	399	185	154	20
47844	Ventana Principal (Abucasis)	2	0	0	1
48761	Menú Calidad	296	111	96	14
49506	Menú Ventana	1	0	1	0
49715	Área de trabajo	7	3	4	0
49776	Menú Servicios Generales	5	2	1	1

Figura 7.9: Vista Tabla Incidencias para el nodo "Ventana principal" generado desde PowerBi

Como se observa en la Figura 7.9, el nodo padre se ha fragmentado en sus nodos hijos, y se ha actualizado tanto la vista gráfica como la vista en forma de tabla, con los datos de los nodos hijos que se están visualizando ahora.

Burbujas como las del nodo 99, nodo 101 y nodo 1043 son aproximadamente del mismo tamaño. Si se observan los datos de la tabla 7.9 se aprecia que el numero total de incidencias de estos nodos son prácticamente iguales (402, 402 y 399 respectivamente) por lo que visualmente les corresponden burbujas del mismo tamaño.

Al igual que el gráfico generado desde Tableau para las pruebas de aceptación es posible observar que mediante el tamaño de las burbujas es bastante intuitivo detectar donde se acumulan las incidencias y poder sacar informes al respecto de que funcionalidades son las que abarcan mayor cantidad de éstas para poder estudiar la causa.

Son informes importantes para ver donde se está fallando o donde es necesario mejorar para tener a los clientes satisfechos.

7.5 Gráfico de barras verticales para la evolución de fallos respecto al tiempo

El último gráfico que se ha incluido en este trabajo es un gráfico de barras verticales donde observar la evolución de las incidencias de fallo en el ultimo año.

Para ello como datos de entrada se parte de dos hojas de cálculo con los valores de los distintos nodos del programa en dos instantes de tiempo:

- La primera hoja de cálculo está generada en Septiembre de 2018 que es cuando empecé a preparar este TFG.

- La segunda hoja de cálculo es el mismo programa a fecha de Junio de 2019.

Este gráfico se ha generado desde Tableau utilizando como datos de entrada las dos hojas de cálculo que se presentaban anteriormente.

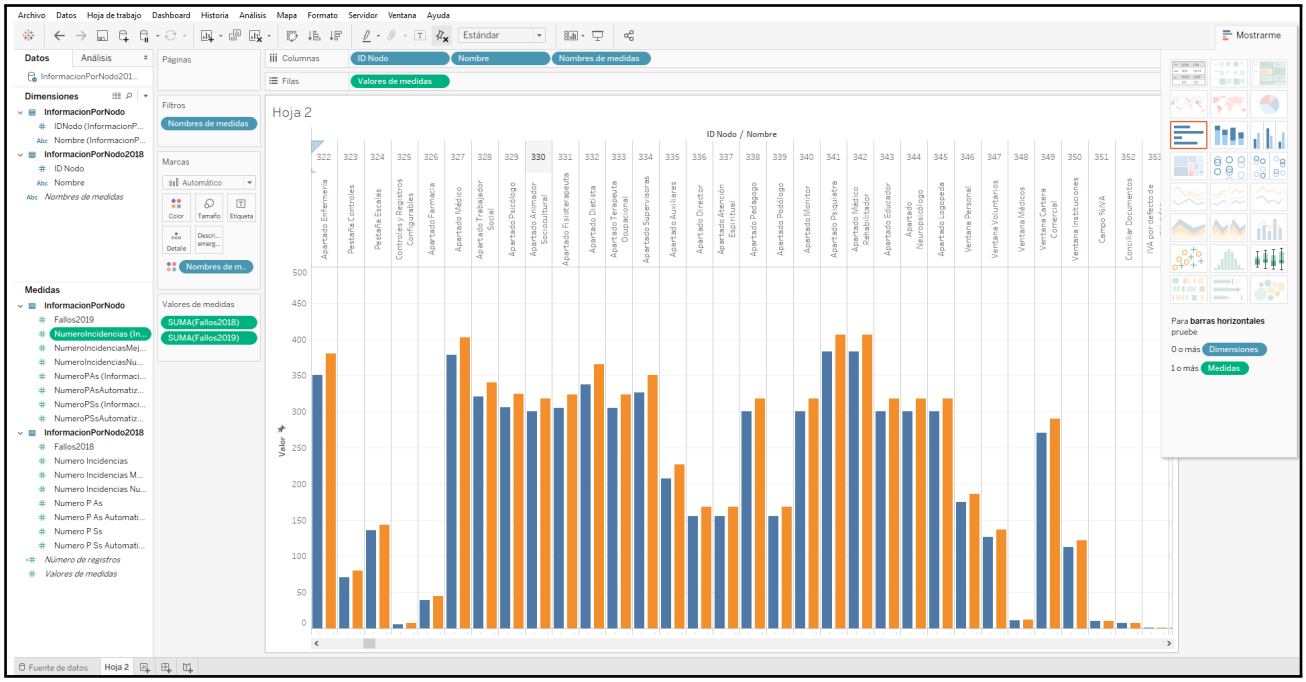


Figura 7.10: Evolución de las incidencias de fallo desde Tableau

En la Figura 7.10 la barra azul corresponde al número de incidencias de fallo a fecha de Septiembre de 2018 y la barra naranja al mismo campo pero en Junio de 2019.

Cada barra o rectángulo representa un nodo de la aplicación de estudio y la altura de la barra indica la frecuencia con la que se presenta ese dato.

Esta herramienta con un solo *click* permite intercambiar filas y columnas y se pasa a visualizar como un gráfico de barras horizontales.

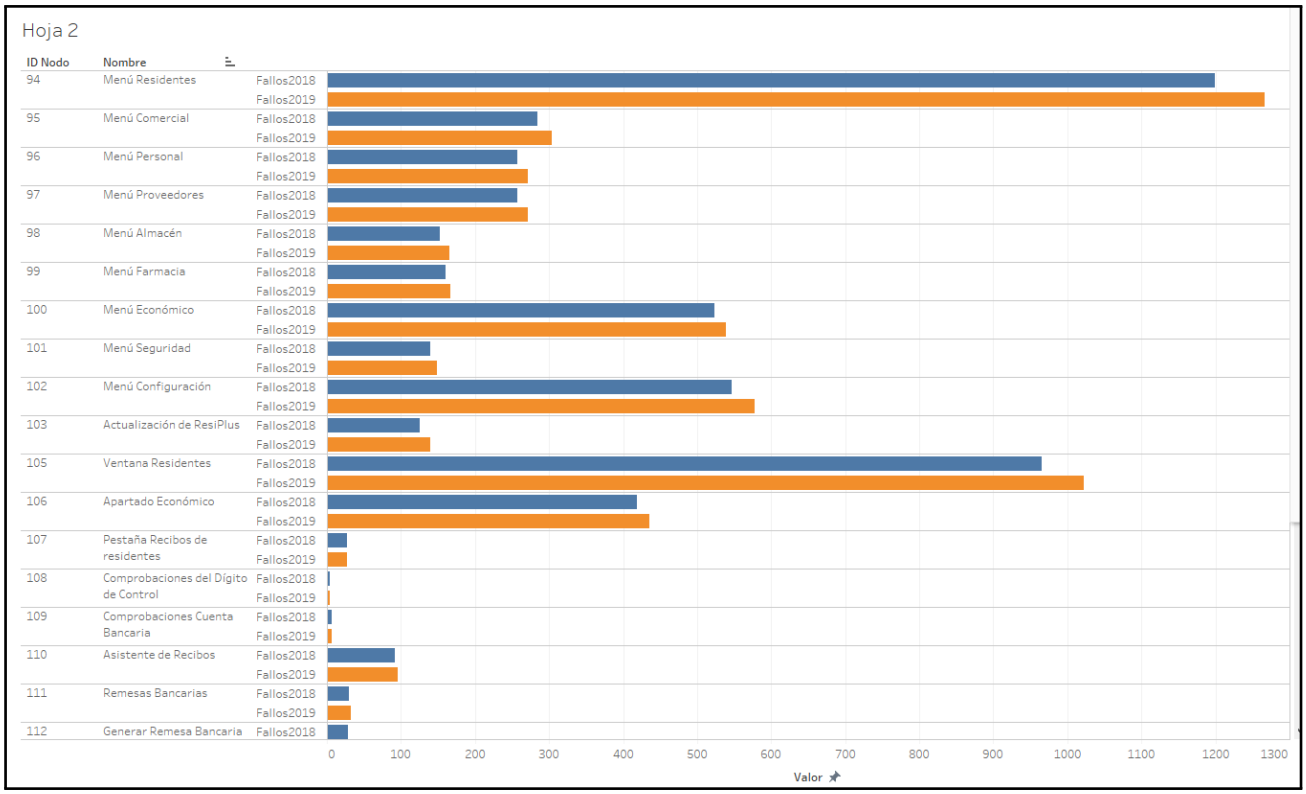


Figura 7.11: Evolución de las incidencias de fallo desde Tableau mediante barras horizontales

Los ejes se adaptan al tamaño del campo que se está midiendo para que la barra no se corte en la visualización.

Colocando el puntero del ratón sobre cualquiera de las barras se abre un *tooltip* donde aparece el nombre del nodo y el numero exacto de incidencias que tiene.

Aquí se puede ver el nodo "Menú Residentes" que ha pasado de 1.199 incidencias de fallo en Septiembre de 2018 a 1.267 en Junio de 2019.

Es posible observar que en la gráfica sería sencillo detectar un crecimiento desmesurado de las incidencias de fallo en cualquier nodo de la aplicación. En este periodo de tiempo que se está estudiando, no se ha observado ningún incremento que sea significativo y alarmante.

Mediante el uso de filtros de la herramienta se podría definir que apareciera primero los nodos donde el crecimiento de fallos ha sido más elevado y así poder estudiar esos casos en más detalle.

CAPÍTULO 8
Comparativa de las distintas herramientas

En este capítulo se van a enumerar las fortalezas y las debilidades que se han encontrado en las distintas herramientas utilizadas para generar las gráficas.

A estas alturas se habrá podido observar que al inicio del estudio se nombraba QlikView entre las herramientas que se iban a analizar pero que no se han incluido gráficas generadas desde la aplicación. Esto es debido a que no se ha logrado una solución totalmente satisfactoria.

QlikView es una herramienta muy básica desde la que fácilmente puedes generar gráficos sencillos. Funciona a través de un asistente que te va guiando paso a paso hasta la visualización del gráfico y la creación del informe.

Cuando se ha intentado generar vistas más complejas en las que entra la herencia de datos no se ha conseguido progresar satisfactoriamente.

A pesar de ser considerada una de las mejores herramientas para el análisis empresarial no se han encontrado tutoriales en la red para poder solventar las dudas y al final se ha desistido de utilizarla.

8.1 Fortalezas y debilidades de Power BI

FORTALEZAS	DEBILIDADES
Es la mas sencilla de utilizar	Solo se pueden compartir las visualizaciones subiendo el proyecto a la nube
Precio mas bajo que el resto	Esta orientada a usuarios avanzados de Excel
Incluye visualizaciones basadas en R	Esta limitada a 10GB de datos
Permite integrar APPs de terceros	No admite archivos de entrada de mas de 250MB
Permite almacenamiento en Azure con todas las ventajas que eso conlleva	
Editor integrado con PowerQuery	
Desarrollada por Microsoft. Integrada con el resto de sus productos	
Muchos tutoriales disponibles en la red	
Permite integración con dispositivos móviles	

Tabla 8.1: Fortalezas y debilidades de PowerBI

8.2 Fortalezas y debilidades de Tableau

FORTALEZAS	DEBILIDADES
Tiene una interfaz visualmente atractiva	Precio mas elevado que sus competidores
Permite la conexión a cientos de orígenes de datos	Necesita una preparación inicial de los datos
Permite la integración con dispositivos móviles	No tiene integración con otras herramientas como Microsoft
Amplio abanico de tutoriales y ofertas de cursos online	Pocas opciones para la transformación de datos
En continuo desarrollo añadiendo nuevas funcionalidades	No permite colaboraciones de terceros
Permite exploraciones mas profundas de los datos	
Tiene soporte para R y Phyton	

Tabla 8.2: Fortalezas y debilidades de Tableau

8.3 Fortalezas y debilidades de JSFiddle

FORTALEZAS	DEBILIDADES
Es totalmente gratuita	Los datos están limitados a una estructura concreta para cada gráfica
No solo soporta JavaScript sino que también HTML o CSS	Esta limitada a unos pocos tipos de gráficos
Fácil de usar	Tiene publicidad emergente
Tiene una API publica	Los errores solo aparecen si utilizas el inspector de código del navegador y son poco intuitivos
Permite descargar los gráficos o compartirlos	

Tabla 8.3: Fortalezas y debilidades de JSFiddle

8.4 Comparativa

Tras exponer los pros y contras de las herramientas utilizadas se va a incluir una comparativa a nivel personal de lo que resaltaría de cada una de las herramientas y en el caso de tener que recomendar alguna a la empresa, por cual me decantaría.

Se han encontrado dificultades tanto en Tableau como en Power BI para realizar la navegación entre los distintos niveles de la estructura en árbol de la aplicación que se ha estado estudiando. Es mas, en ninguna de las dos herramientas se ha conseguido que el explosionado de las burbujas fragmentándose en las que están en el nivel inmediatamente inferior sea automático. En cambio, desde la herramienta Web esto si que se produce.

En Power BI se ha solventado mediante el uso de un filtro de terceros que representa la herencia y que mediante unos sencillos pasos permite seleccionar que nodo quieres que explosione.

Power BI tiene un *MarketPlace* donde otros desarrolladores suben soluciones a los problemas que se han ido encontrando y solventando. Es gratuito y se descarga en segundos como un *pluggin* de la aplicación de escritorio.

Desde Tableau ha sido algo mas complejo poder hacer explosionar las burbujas porque ha habido que definir campos calculados y parámetros adicionales. Aun así, es más complicado de utilizar este mecanismo que el definido en la herramienta competidora.

Power BI gana a la competencia en cuanto a facilidades para integrar software de terceros puesto que permite colaboraciones de desarrolladores cosa que Tableau no tiene integrada, tiene una amplia comunidad pero solo para dudas y consultas.

Otro aspecto a comparar es el precio de adquisición de ambas herramientas. Tableau está más orientada a expertos en el análisis de datos y su precio es bastante más elevado. Power BI está dirigida a todo tipo de usuarios que necesitan del BI para mejorar sus análisis, y por eso, su precio básico es más competitivo.

JSFiddle es gratuita y ofrece visualizaciones atractivas tal y como el gráfico de proyección solar que se ha incluido en el capítulo anterior pero no puede competir con estas otras herramientas en cuanto a calidades, prestaciones y alcance. Está bien para generar alguna gráfica concreta pero no cumple los propósitos perseguidos por este TFG.

Tras utilizar ambas herramientas y de acuerdo al objetivo de este trabajo recomendaría Power BI por delante de Tableau porque me ha parecido más fácil de usar, está integrada con Microsoft, tiene mucha documentación y tutoriales en la red, su precio es más asequible y cumple los objetivos de poder generar informes y analizar el producto.

Tableau es una herramienta más completa, más profesional, más cara y más compleja de usar. Está más orientada a expertos en *big data* y ello sobrepasa en creces las necesidades reales de esta empresa y en concreto de este estudio.

CAPÍTULO 9
Conclusiones

Cada vez son más las empresas que disponen de un volumen insospechado de datos de los cuales no se saca rendimiento.

John Naisbitt se refiere a esto diciendo:

"We are drowning in information, but starving for knowledge." ("nos ahogamos en información, pero estamos hambrientos de saber") [7].

En el mercado existen soluciones que ofrecen un análisis sencillo de tus datos, sin necesidad de transformaciones ni de saber programar, pero al final en este trabajo se ha comprobado que no siempre resulta ser así.

En el desarrollo de este trabajo se ha podido observar que dichas herramientas si que son sencillas de utilizar pero solo una vez que se tienen todos los datos estructurados de acuerdo a unos patrones necesarios por la herramienta en cuestión.

Resaltar que no se ha encontrado una solución satisfactoria al objetivo perseguido por este trabajo ya que no se ha conseguido una navegación automática entre los distintos niveles de las gráficas, es decir, un explosionado automático de las burbujas en el caso del gráfico de burbujas y una fragmentación automática de los anillos en el caso del gráfico *sunburst*.

Con este trabajo se ha logrado:

- Estructurar la gran cantidad de datos que había en la empresa y poder generar conocimiento de la aplicación.
- Mejorar y corregir los cálculos que se mostraban en la herramienta de gestión interna (SAPI).
- Identificar las gráficas necesarias para visualizar los datos jerárquicos de la empresa, en concreto, gráfica de burbujas y gráfica *sunburst*.
- Crear gráficos dinámicos y visualizaciones atractivas para ayudar a la toma de decisiones.
- Probar y analizar las distintas herramientas disponibles en el mercado para generar información de una manera sencilla.

Implantar el *Big Data* en una empresa es una tarea larga que requiere de la predisposición de cada uno de los equipos que la conforman, a pesar de ello con este TFG se han dado los primeros pasos para poder generar conocimiento desde diferentes fuentes de entrada de datos.

Como punto de vista personal este TFG me ha permitido trabajar en un proyecto real de la empresa e investigar y generar gráficas con herramientas muy utilizadas en el BI. Los conocimientos adquiridos en los estudios me han capacitado para enfrentarme con éxito a los problemas que se me han ido planteando en el transcurso de este trabajo, por ejemplo, lo aprendido en "Estructuras de Datos y Algoritmos (EDA)" me ha servido de base para poder transformar el grafo dirigido en un

árbol mediante recursividad o lo aprendido en "Bases de Datos y Sistemas de Información (BDA)" para realizar consultas SQL e incluso montar un servidor propio de base de datos.

Como trabajo futuro se proponen las siguientes tareas:

- Programar una rutina que tras cada cierre de versión actualizará la hoja de cálculo donde se almacenan los datos del programa.
- Mejorar la generación de los datos en el formato que permita exportarlos a otras herramientas.
- Validar y detallar con los analistas las visualizaciones de datos.
- Automatizar la generación de estos gráficos y crear informes mensuales que se distribuyeran de manera automática entre el equipo de desarrollo.

Bibliografía

[1] Patricio Letelier Desarrollo ágil basado en pruebas de aceptación *http://iwt2.org/wp-content/uploads/2015/05/Ponente-5-ISSI-Universidad-P.-Valencia.pdf*, UPV 2015.

[2] Company, M., Letelier, P., Marante, M., Suarez, F. Análisis de impacto en requisitos soportado de forma semi-automática y en un marco de desarrollo TDD basado en pruebas de aceptación *XVI Jornadas de Ingeniería de Software y Bases de Datos (JISBD)*, 2011.

[3] IBM Watson Marketing. Marketing in the Dark - Dark Data *Econsultancy*, Enero de 2018.

[4] Pigoski, Thomas M. *Practical Software Maintenance*. Nueva York: John Wiley & Sons. ISBN 978-0-471-17001-3.

[5] Sommerville, Ian *Software Engineering*. Universidad de Michigan: Addison-Wesley, 2005. ISBN 978-0-201-56529-4.

[6] MCclure, Carma. *CASE la automatización del software*. España: Ra-Ma, 1992. ISBN 84-7897-055-X.

[7] Naisbitt, John. *Megatrends: Ten New Directions Transforming Our Lives*. EEUU: Grand Central Publishing, 1988. ISBN 978-0446512510.

[8] Definición del concepto de Big Data. https://www.oracle.com/es/big-data/guide/what-is-big-data.html, Consultado en Julio de 2019.

[9] Definición del concepto de RMF. https://en.wikipedia.org/wiki/Requirements_Modeling_Framework, Consultado en Julio de 2019.

[10] Definición de FMEA. http://www.fmea-fmeca.com/fmea-rpn.html, Consultado en Julio de 2019.

[11] Página oficial de UML. Definición. https://www.uml.org/what-is-uml.htm, Consultado en Julio de 2019.

[12] Documentación oficial de Entity Framework. https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/ef/overview, Consultado en Julio de 2019.

[13] Documentación sobre el lenguaje LinQ. https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/working-with-linq, Consultado en Julio de 2019.

[14] Definición de SaaS. https://azure.microsoft.com/es-es/overview/what-is-saas/, Consultado en Julio de 2019.

[15] Definición de ORM. https://en.wikipedia.org/wiki/Object-relational_mapping, Consultado en Julio de 2019.

[16] Pagina web de la documentación de Microsoft para la estructura de datos Dictionary. https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.8, Consultado en Julio de 2019.

[17] Definición de Business Intelligence. https://es.wikipedia.org/wiki/Inteligencia_empresarial, Consultado en Julio de 2019.

[18] Informe completo del Cuadrante Mágico de Gartner para BI. https://www.gartner.com/doc/reprints?id=1-3TXXSLV&ct=170221&st=sb, Consultado en Julio de 2019.

[19] Pagina web de la documentación de Microsoft para XML y DOM. https://docs.microsoft.com/en-us/dotnet/standard/data/xml/xml-document-object-model-dom, Consultado en Julio de 2019.

[20] Pagina web de la documentación de JSON. https://www.json.org/json-es.html, Consultado en Julio de 2019.

[21] Pagina web de la documentación de la APP Hierarchy Slicer de PowerBi. https://appsource.microsoft.com/en-us/product/power-bi-visuals/WA104380820?tab=Overview, Consultado en Julio de 2019.