Universidad ORT Uruguay

Facultad de Ingeniería

GDR – Sistema Generador de Reportes para sistemas de gestión

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Federico Agripa – 150177

Diego Barral – 150178

Joaquín Colella – 164478

Tutor: Santiago Matalonga

2016

Declaración de autoría

Nosotros, Federico Agripa, Diego Barral y Joaquín Colella, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto de grado de fin de curso de la carrera Ingeniería en Sistemas;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa entra trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Federico Agripa

Diego Barral

Joaquín Colella

3 de marzo de 2016

Agradecimientos

De AT&G Informática: Gerardo Taube, por su predisposición, disponibilidad y compromiso con el equipo.

De Universidad ORT Uruguay: A nuestro tutor, Santiago Matalonga, por el constante apoyo al grupo y disponibilidad tanto dentro como fuera del horario laboral. A Juan Emilio Gabito y Nicolás Fornaro por el asesoramiento acerca de diversas tecnologías.

A nuestras familias y amigos por el apoyo durante el año que duró el proyecto.

Abstract

AT&G Informática es una empresa de *software* cuyo principal producto es PSIG ERP, un *software* de gestión empresarial. Dicho *software* carece de un módulo que le permita al usuario obtener reportes a partir de sus datos. Anteriormente, la empresa había experimentado con el uso de herramientas de *business intelligence*, con resultados insatisfactorios tanto para el cliente como para el usuario.

Es así que se identificó la necesidad de una solución que permita al usuario obtener reportes útiles para la toma de decisiones empresariales en tiempo real en base a los datos que el sistema PSIG ERP genera.

Para esto se investigaron tres diferentes tecnologías de bases de datos, las cuales se especializan en mejorar los tiempos de selección de información. Una vez seleccionada la mejor opción, se procedió a construir un sistema web fácil de usar que permita al usuario obtener cualquier reporte que desee.

Luego de considerar diferentes alternativas de cómo implementar el sistema, se llegó al resultado final que consiste en tres módulos. Un módulo que se encarga de recibir los cambios en la base de datos de PSIG y replicarlos en la base de datos de nuestro sistema. Un módulo web que es con el que interactúa el usuario, y por último un módulo que ejecuta las consultas que el usuario necesita sobre nuestra base de datos.

El sistema se evaluó con usuarios reales y está disponible para ser instalado en un cliente real, las conclusiones de estas pruebas pueden evidenciarse con mayor detalle en la sección de conclusiones.

En concreto todavía no fue instalado ya que se están desarrollando particularidades para satisfacer necesidades específicas de un cliente con altas expectativas definido por AT&G Informática.

Palabras clave

ERP; **UI; FDD; Reporte; Groovy on Grails**

Glosario

API: es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. [1]

Bases de datos NoSql: Bases de datos que utilizan un lenguaje de consultas distinto a SQL.

***Brainstorming*:** herramienta de trabajo grupal que facilita el surgimiento de nuevas ideas sobre un tema o problema determinado. [2]

Cubo: base de datos multidimensional, en la cual el almacenamiento físico de los datos se realiza en un vector multidimensional. [3]

***Data Mining*:** es un campo de las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos. [4]

***Data Warehouse*:** colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza. [5]

Deployment: actividades que hacen un sistema de software disponible para su uso. [6]

ERP: Sistema de planificación de recursos empresariales.

FDD: *Feature Driven Development*.

***Feedback*:** información acerca de la reacción sobre un producto usado como base para la mejora.

Interesados: entidades que tienen interés en un proyecto dado. [7]

JSON: formato de intercambio de datos liviano fácil para los humanos de leer y escribir, y fácil para las computadoras para analizar y generar. [8]

***Middleware*:** software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. [9]

Minar: Proceso mediante el cual un usuario obtiene información a partir de un *data warehouse*.

POC: *Proof of Concept*

Reporte: Información obtenida a partir de datos que aporta valor para quien realiza una consulta.

RESTful API: API con estilo arquitectónico REST.

SCM: *Software Configuration Management.*

SOAP: protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. [10]

SWBOK: *Software Engeneering Body of Knowledge*.

UI: Interfaz de usuario.

Índice

1 Introducción

El objetivo de éste documento es describir el proyecto “GDR - Sistema generador de reportes para sistemas de gestión”, realizado como requisito para la obtención del título Ingeniero de Sistemas en la Universidad ORT Uruguay.

El proyecto consiste en crear un *middleware* y una aplicación web orientado a empresas que permita al usuario cruzar información de sus clientes almacenada en una base de datos, con el fin de crear reportes que faciliten la toma de decisiones empresariales en tiempo real. El producto se diferencia en que permite crear nuevos reportes a gusto del usuario mediante una interfaz intuitiva y fácil de usar, y en márgenes de tiempo sensiblemente menores a los actuales.

Entonces a efectos prácticos, el producto del proyecto consiste en una aplicación web que permita a los usuarios crear nuevos reportes y otras funcionalidades asociadas a ellos, acoplada a un sistema que permita interpretar lo que el usuario quiere, y obtener la información necesaria de la base de datos. Por otro lado, deberá crearse un sistema que obtenga la información necesaria para cargar la base de datos, y luego inserte la información obtenida. El núcleo del sistema será una base de datos que sea capaz de manejar grandes volúmenes de datos y pueda ejecutar consultas sobre millones de registros en fracciones de segundo.

El proyecto surge a partir de la identificación de la necesidad de la empresa AT&G Informática, de un sistema de reportes para su principal producto, PSIG ERP. AT&G Informática es una empresa de *software* que comercializa sistemas de gestión empresarial. PSIG ERP es su principal producto y permite tanto gestionar los recursos de una empresa como manejar las finanzas y registros de compras y ventas, por lo tanto cuenta con toda la información de la empresa. Su principal carencia era la falta de un módulo que le permitiera obtener información a partir de los datos almacenados, lo cual haría el producto más atractivo a potenciales clientes.

En el pasado, AT&G Informática había experimentado con diversas herramientas de *reporting*, nunca obteniendo resultados satisfactorios. El equipo entonces le planteó a la empresa la idea de crear un sistema que le permitiera crear reportes sobre sus datos y desarrollado con *software* libre.

1.1 Objetivos

1.1.1 Objetivos del producto

Generar información para los usuarios del producto en base a los datos que estos manejan de forma cotidiana y no son capaces de aprovechar con las herramientas que cuentan actualmente. Con esto se busca abastecer al usuario de información que le permita conocer objetivamente datos que pueden ser útiles tanto para la toma de decisiones como para predicciones futuras.

1.1.2 Objetivos del proyecto

Lograr consolidar un equipo de trabajo, capacitarnos en nuevas tecnologías y áreas de conocimiento, ya sea informáticas o de otros rubros. Crear un producto de calidad que cubra la necesidad de AT&G Informática, y agregue valor para los usuarios.

1.1.3 Objetivos académicos

El principal objetivo académico del equipo fue obtener el título de Ingeniero de Sistemas, lo cual representó una gran motivación.

A lo largo de la carrera, al equipo le fueron provistas de diversas técnicas y herramientas que permitieron llevar adelante un proyecto de *software* de principio a fin. Por lo tanto, fue para el equipo un objetivo aplicar todo lo estudiado en la carrera en éste proyecto, obtener un producto que pudiese ser utilizado para lo que se creó y que generase satisfacción en sus usuarios.

Por último, debido a la constante evolución del mundo del *software*, fue para el equipo una prioridad aprender acerca de nuevas tecnologías, para así mantenerse actualizado con las tendencias actuales. Es por esto que en éste proyecto se usaron tecnologías que el equipo no estudió durante la carrera pero que son muy usadas en la industria.

1.2 Equipo

Se conformó un equipo multidisciplinario y altamente motivado, con miembros con experiencia tanto en base de datos como en desarrollo web. A pesar de que los miembros de

este equipo nunca habían trabajado juntos anteriormente, fue clave la buena comunicación a lo largo del proyecto.

2 Descripción del proyecto

Con el objetivo de brindar una descripción de alto nivel del desarrollo del proyecto, este capítulo describe las principales características del mismo.

2.1 Características del proyecto

Participación del cliente

El equipo contó con un cliente muy participativo en el seguimiento del proyecto, proporcionando *feedback* constantemente. Esto permitió detectar y gestionar insatisfacciones por parte del mismo.

Otro factor determinante para la evolución del proyecto fue la confianza que el cliente depositó en el equipo permitiendo que este tomará las decisiones. Debiendo respetar las necesidades del cliente en todo momento.

Utilización de nuevas tecnologías

Acompañando los intereses de los integrantes del equipo, se decidió utilizar tecnologías no vistas durante la carrera para el desarrollo del proyecto, con el fin de crecer a nivel profesional. Esta decisión generó un aumento en la carga horaria de gestión, debiendo minimizar el impacto negativo que genera una curva de aprendizaje de nuevas tecnologías.

2.2 Distribución del proyecto

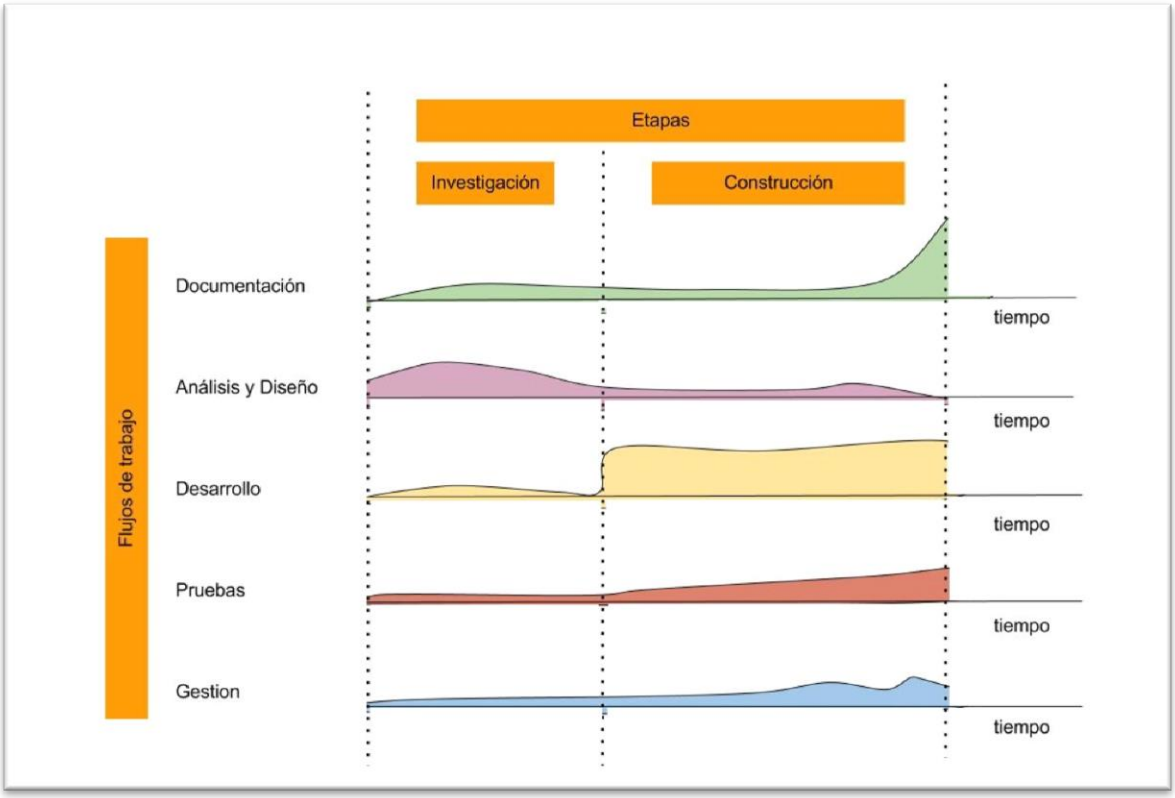

Figura 2-1: diagrama de distribución del esfuerzo

En el diagrama anterior se observa como fue la distribución del esfuerzo en una línea de tiempo que abarca desde el comienzo del proyecto hasta el fin del mismo. La misma está divida en dos etapas que son las que dividen al proyecto. Esta división se da con el comienzo de la iteración 1, la cual da lugar al inicio de la etapa de construcción y fin a la etapa de investigación. El esfuerzo puede verse desglosado por los flujos de trabajo que contiene el proyecto.

La primer etapa, de investigación, es todo lo realizado por el equipo desde el comienzo del proyecto para poder adquirir el conocimiento y los elementos necesarios para comenzar con la construcción del sistema. La misma culmina con el comienzo de la construcción, que como se menciona al principio, se da con el comienzo de la primer iteración del desarrollo.

La segunda y última etapa es la construcción, esta abarca tanto el mayor esfuerzo del proyecto (en cantidad de horas invertidas en la misma), como la mayor duración. La misma tuvo inicio el 20 de Julio de 2015.

El primer flujo de trabajo que podemos encontrar en el diagrama es la documentación, el esfuerzo invertido en esta se mantuvo estable en la mayor parte del proyecto, teniendo un aumento en los comienzos del proyecto al documentar todo lo vinculado a pruebas de concepto y análisis de tecnologías. Sobre el final del mismo, la documentación tuvo un aumento relacionado a la proximidad del final del proyecto y a la necesidad del equipo de tener que organizar, revisar y mejorar versiones del presente trabajo que se terminaría entregando.

En cuanto al análisis y diseño, este flujo de trabajo se encuentra elevado en la primer etapa del proyecto, donde se debía analizar tecnologías, diseñar posibles soluciones, así como también diseñar pruebas de concepto. Durante la etapa de construcción el análisis y diseño tuvo un aumento en el esfuerzo dedicado cuando el equipo detectó un problema en la tecnología aplicada a la base de datos, que implicó un cambio de solución que derivó en tener que analizar y diseñar la nueva solución a aplicar para suplantar la que produjo problemas.

El desarrollo es un flujo de trabajo invertido principalmente en la etapa de construcción, en esta fue donde se construyó el sistema basado en el desarrollo del mismo. A pesar de esto, al principio del proyecto estuvo también presente este flujo, cuando el equipo tuvo que desarrollar soluciones a tecnologías para poder realizar pruebas de concepto sobre las mismas, pudiendo comparar resultados.

Sobre las pruebas se puede decir que las mismas aumentaron llegando al final del proyecto, haciendo que el equipo durante este tiempo empiece a focalizarse más en el correcto funcionamiento del sistema. Previo a esto la inversión de esfuerzo en pruebas era constante.

Por último la gestión es un flujo de trabajo al que el equipo le invirtió esfuerzo constante durante todo el proyecto, habiendo aumentado el mismo en dos ocasiones puntuales en la etapa de construcción. Estas ocasiones fueron problemas que enfrentó el equipo, que implicaron re planificaciones en el proyecto, tomando la gestión un papel más importante a la hora de administrar los recursos disponibles de forma que estos cambios afecten lo menos posible al resto de actividades del proyecto.

2.2.1 Distribución horaria

En cuanto a la cantidad de horas total invertidas en el proyecto por el equipo fue de 2446 horas. Pudiendo desglosar las mismas según flujo de trabajo:

- 1003 horas de desarrollo conformado por 860 horas en etapa de construcción y 143 horas en etapa de investigación.
- 424 horas de análisis y diseño.
- 271 horas de prueba, dentro de prueba no se consideran las pruebas del desarrollador que están incluidas dentro de las horas de desarrollo del mismo.
- 492 horas de documentación.
- 256 horas de gestión.

3 Requerimientos del *software*

El presente capítulo describe todo lo relativo a los requerimientos del sistema. Se define qué es un requerimiento, los distintos tipos de requerimientos, los requerimientos en sí mismos y como se obtuvieron.

3.1 Fundamentos de los requerimientos

Requerimientos de producto y proceso

En esta sección se describen los principales requisitos del producto y proceso. De acuerdo al SWBOK, se interpreta como requisitos del producto como "una necesidad o restricción de *software* a ser desarrollada", y requisito del proceso como "una restricción del desarrollo del *software*". [11]

Para obtener los requerimientos del sistema, se planteó inicialmente un conjunto de necesidades iniciales propuestas por el equipo y se fue refinando. Esto fue así debido a que el cliente nos solicitó que nosotros propusiéramos una solución que se adaptara a sus necesidades. Las etapas tempranas de relevamiento y definición de los requerimientos fueron llevadas a cabo principalmente por el equipo y posteriormente conllevó una mayor carga horaria validar los mismos.

3.2 Requerimientos funcionales y no funcionales

A continuación se presenta una lista de los requerimientos funcionales y no funcionales. Los requerimientos funcionales fueron escritos en forma de *feature list*; el concepto de *feature list* y por qué los requerimientos fueron especificados de esta forma se encuentra en el capítulo Metodología de Trabajo (para más detalle sobre los mismos ver el anexo de Requerimientos funcionales).

La siguiente es la lista de *features* acordados con el cliente en primera instancia:

- Configurar reporte
- Traducir consulta
- Guardar reporte localmente como archivo
- Guardar reporte en el sistema

- Reportes favoritos
- Modificar reportes
- Ver reporte favorito
- Generación de reportes planificados
- Manejo de usuarios
- Privilegios
- Recibir datos de PSIG
- Procesar e insertar datos en la base de datos
- Sesión
- Solicitud de permisos
- Enviar reporte por mail
- Generar reporte a partir de otro
- Log de actividad
- Ejecución de consulta

En el siguiente listado, se presentan los *features* que surgieron a medida que avanzó el proyecto, y que por consiguiente, modificaron el alcance:

- Cambiar base de datos en columnas a Infobright
- Cambiar interfaz gráfica del usuario
- Compartir reporte

Vale aclarar que a partir del requerimiento "Recibir datos de PSIG" se deriva la necesidad de desarrollar una interfaz. Las necesidades y motivo de esta interfaz serán presentadas en el capítulo "Arquitectura" y "Construcción".

A continuación se presenta una lista de los requerimientos no funcionales:

- Un reporte debe poder obtenerse en menos de un minuto independientemente del volumen de datos
- El producto deberá soportar hasta 20 usuarios trabajando en simultaneo sin experimentar *downgrades*
- El producto que utilicen los operadores debe ser intuitivo para los mismos
- El sistema debe correr en Windows Server 2008 (o similar) o posterior
- El sistema permitirá generar reportes de Ventas o Compras

- El sistema debe contemplar la posibilidad de incorporar nuevas entidades al sistema además de compras o ventas, por lo que no deberá ser difícil modificar los datos con los que trabaja.

3.2.1 Propiedades emergentes

El rendimiento deseado en la velocidad de generación de un reporte con los últimos datos disponibles por el sistema PSIG ERP depende de dos factores clave:

- La frecuencia con la que PSIG nos envíe los datos
- La frecuencia con la que podamos actualizar la base de datos en columnas (nuestro *data warehouse*)

3.2.2 Requerimientos cuantificables

Si bien todos los requerimientos deben ser cuantificables, se hizo un énfasis especial en algunos requerimientos en concreto, dada la importancia de los mismos en el proyecto/producto: usabilidad y rendimiento.

Usabilidad:

- El usuario con una explicación acotada de no más de 30 minutos debe ser capaz de utilizar el sistema de forma intuitiva.
- El usuario siempre debe tener visible de forma clara la acción que esté realizando

Rendimiento:

- El sistema tendrá actualizados los datos que se le envíen en un máximo de 2 horas.
- La generación de un reporte no debe superar un minuto

3.3 Proceso de elicitación de requerimientos

Para refinar los requisitos se siguió el siguiente proceso.

3.3.1 Modelo

Se partió de un conjunto de requerimientos que planteó el equipo, y mediante entrevistas con el cliente se fueron refinando a la vez que surgían nuevos. Estas entrevistas fueron realizadas

de forma desestructurada, y a medida que se comentaban los requerimientos iniciales se discutía posibles casos de uso que no fueron contemplados en etapas anteriores.

Debido a las grandes expectativas en cuanto a usabilidad del cliente, se realizaron prototipos de baja definición para validar aspectos de la interfaz de usuario, y así alinear las ideas del cliente con las del equipo. Una vez que se llegó a un conjunto formal de requerimientos acordado por ambas partes, se procedió a especificar los mismos en un documento de requerimientos del sistema (presente en el anexo Requerimientos Funcionales).

A medida que el proyecto avanzó, se realizaron validaciones sobre interfaces de usuarios ya funcionales. Varios cambios surgieron de dichas validaciones, que siempre resultaron en mejoras a la usabilidad.

También se hicieron validaciones con personas que gerentes de empresas y como tales, les interesa tener información sobre su negocio, por lo tanto una herramienta de ésta naturaleza les sería útil. Dichas instancias resultaron en *feedback* útil para el equipo, que fue tenido en cuenta para las siguientes iteraciones de la aplicación.

3.3.2 Interesados

Los interesados definidos a continuación fueron identificados en conjunto entre el equipo y nuestro principal cliente, Gerardo Taube, quién nos indicó quienes serían los posibles usuarios sistema dentro de una empresa.

Usuarios

Los usuarios del sistema serán gerentes de empresas, altos cargos administrativos y otros cargos relativos a la gestión de la empresa. Estos usuarios no necesariamente deben tener conocimientos informáticos, esto es, capacitación específica acerca de tecnologías informáticas.

Clientes

Nuestro cliente principal es AT&G Informática, e indirectamente toda su cartera de clientes que serían quienes en última instancia usarían el producto.

Desarrolladores

El equipo compuesto por Federico Agripa, Diego Barral y Joaquín Colella.

3.3.3 Fuentes de requerimientos

El SWBOK define un conjunto de fuentes de requerimientos para cualquier proyecto de *software*. A continuación se indica cuáles y cómo fueron utilizadas en nuestro proyecto.

- **Objetivos**: El equipo planteó objetivos para crear un producto enfocado en las necesidades y falencias del mercado previamente descrito al principio del documento.
- **Conocimiento del dominio**: El equipo consideró que para poder desarrollar cualquier producto de *software* con un alto nivel de calidad, entender los requerimientos desde el punto de vista de un experto del dominio es clave, y el pilar fundamental de las futuras decisiones que deban tomarse.
- **Interesados**: La correcta detección y ponderación de los intereses de cada uno de los interesados permite gestionarlos y evitar en etapas tardías tener que optar por la satisfacción de unos u otros.
- **Objetivos del negocio**: Sirvieron como criterios de inclusión o exclusión en los factores que luego compondrían el comportamiento del sistema en sí mismo.
- **Entorno de operaciones**: el equipo decidió enfocarse principalmente en la usabilidad y performance del sistema, ya que para poder competir con herramientas similares que actualmente rigen en el mercado, los usuarios deben percibir que el sistema sea más fácil de usar y más rápido que las mismas.
- **Entorno organizacional**: El equipo debió adaptar algunos aspectos del sistema debido a la cultura organizacional del cliente y su estructura.

3.4 Herramientas para documentar requerimientos de *software*

Para documentar las necesidades del cliente surgidas en las reuniones, se utilizaron minutas de reunión (las mismas pueden encontrarse en el anexo Evidencias Digitales), debido a que desde el inicio ya estaba bastante claro qué debía hacer el sistema, pero restaba incorporar e inferir algunas de las necesidades del cliente.

3.5 Técnicas de relevamiento

De las técnicas de relevamiento de requerimientos presentes en el SWBOK [11], se explican a continuación las que fueron aplicadas en el proyecto.

Entrevistas

Mediante las mismas se pudo acotar el alcance inicial del producto y definir quiénes serían los actores principales. El equipo aplicó ésta técnica durante la etapa de investigación para obtener requerimientos que el cliente necesitaba pero que el equipo no fue capaz de identificar al principio.

Escenarios

Forman parte de la esencia de FDD al momento de definir *feature lists.* Se plantean casos reales de uso y se extraen *features*.

Prototipos

Los prototipos sirven para mostrar ejemplos parcialmente funcionales para obtener *feedback* del cliente. Se utilizaron en dos etapas del proyecto. Inicialmente para probar la viabilidad de los requerimientos y que el cliente pudiera experimentar los tiempos de respuesta que se deseaban plantear. Luego fueron utilizados prototipos para relevar características de usabilidad.

Observación

Mediante la observación de sistemas ERP actuales y otros sistemas de generación de reportes se pudo obtener cuales eran los requerimientos clave a la hora de construir el sistema. Cuales requerimientos existentes los usuarios valoran más y cuáles son los que desean y no poseen.

Objetivos

Mediante la observación de otros sistemas y las necesidades latentes de los usuarios de sistemas de *reporting*, el equipo pudo definir requerimientos basados en objetivos a largo plazo que se orientaban en satisfacer necesidades del negocio.

Brainstorming

El equipo se reunió para generar un listado de funcionalidades que el sistema debería realizar.

4 Metodología de trabajo

Éste capítulo tiene como objetivo explicar qué metodología de trabajo utilizó el equipo durante el proyecto, por qué y cómo la aplicamos en la práctica. El capítulo comienza describiendo particularidades del proyecto, las distintas opciones de metodologías existentes, la decisión que se tomó y por qué fue la que mejor se adecuaba al proyecto.

4.1 Particularidades del proyecto

La metodología que se usa en un proyecto tiene que decidirse en base a las características del proyecto en el que se está trabajando. Quiere decir que hay metodologías que permiten explotar de mejor manera las características de un proyecto que otras.

El proyecto contó con dos factores que fueron determinantes al momento de tomar la decisión. El primero y principal factor fue la estabilidad de los requerimientos. Debido a que el proyecto surge como una propuesta del equipo, el cliente solicitó que el equipo propusiera un conjunto de requerimientos inicial. Luego de un par de instancias de negociación, y con algunos requerimientos sugeridos por el cliente, se llegó a un acuerdo de un conjunto de requerimientos iniciales, por lo tanto, el equipo tenía un alcance fijo.

El segundo factor fue la necesidad de proveer *feedback* frecuente al cliente, ya que tanto él como el equipo, quería validar los avances a medida que se desarrollaba el producto.

Es por esto que el equipo decidió que la metodología de trabajo debía ser derivada del manifiesto ágil, y en concreto, la que más se adaptaba a las condiciones del proyecto era *Feature Driven Development*.

4.2 Justificación de la selección

Feature Driven Development fue inicialmente concebido por Peter Coad y sus colegas [12] como un modelo de proceso práctico para la ingeniería de *software* orientado a objetos. Luego Stephen Palmer y John Felsing [13] extendieron y mejoraron lo hecho por Coad, describiendo un proceso ágil y adaptativo que puede ser aplicado a proyectos de *software* de tamaño moderado y grandes.

En el contexto de *Feature Driven Development*, un *feature* es "*una funcionalidad que tiene valor para el cliente y que puede ser implementada en dos semanas o menos*" [12]. La definición de *feature* trae consigo ciertos beneficios [14]:

- Entregar *software* operacional cada dos semanas
- Los *features* pueden ser agrupados jerárquicamente
- La planificación y seguimiento se hacen según la jerarquía de *features*

El proceso de obtención de *features* y construcción puede ser representado por el siguiente diagrama:

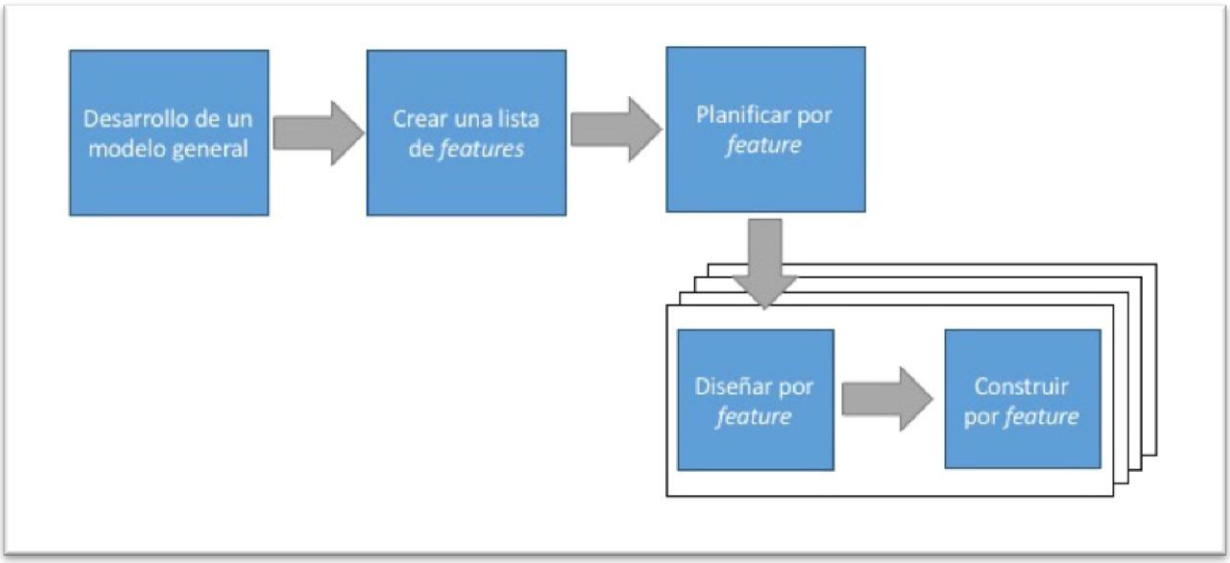

Figura 4-1: proceso de *Feature Driven Development*

Desarrollo de un modelo general

Consiste en crear un conjunto de *features* básicos y un modelo arquitectónico base, que sirve como planteo inicial. Le da forma al proyecto. En el proyecto, ésta etapa consistió en plantearle un conjunto inicial de requerimientos al cliente junto con una idea global de cómo se plantearía la arquitectura del sistema.

Crear una lista de *features*

Se crea una lista de *features* que surgen a partir de los sugeridos en la etapa anterior. Se agrupan en *feature sets* y se mapean con componentes arquitectónicos. En ésta etapa, se

realizaron varias reuniones con el cliente para definir y validar cuál sería la lista final de requerimientos y acordar cómo sería la estructura arquitectónica.

Planificar por *feature*

Se estima el tamaño de cada *feature*, se le da un nivel de prioridad, y se planifica en base a ello. Para estimar el tamaño de cada *feature*, el equipo los comparó entre ellos, decidiendo cuál tomaría más tiempo para desarrollar. Para aquellos que se pensara que fuera muy grande, se partían en dos o más *features* que fueran abarcables en una iteración. Luego, dependiendo la importancia que el *feature* tuviera para el cliente, se le asignó una prioridad, siempre respetando la jerarquía y precedencia de los mismos.

Diseñar por *feature*

Para cada *feature* de la iteración, se diseña la implementación que se seguirá para implementarlo. El equipo realizó reuniones al comienzo de cada iteración para discutir cómo se implementarían los *features* correspondientes.

Construir por *feature*

Se construye el *feature* generando valor para el cliente. Ésta etapa tiene al final una instancia de pruebas, inspección de código e integración en ese orden.

Una de las principales características de *Feature Driven Development* es que la mayor parte de los requerimientos deben ser conocidos de antemano y son estables. Además hay que contar con una arquitectura mínima definida, a la que se puedan mapear los *features* planteados.

Comparación con otros modelos de gestión

El equipo decidió descartar las metodologías tradicionales ya que los principios del manifiesto ágil [15] hacen foco, entre otras cosas, en la calidad del producto final, y dan una gran importancia a la participación del cliente durante el proceso. Al equipo contar con la predisposición del cliente de formar parte del proceso e inclusive este solicitar avances se decantó en las metodologías ágiles ya que satisfacían de mejor manera la realidad del proyecto.

Dentro del mundo ágil la metodología más popular ha sido Scrum desde hace años, pero se comparó en concreto con FDD que parecía tener factores que favorecieran mejor las necesidades y realidad del proyecto GDR.

A continuación se presenta una tabla comparando características de cada metodología y como estas impactan en el proyecto.

Característica	**FDD**	**Scrum**	**GDR**
Definición de actividades	Basado en procesos	Actividades de alto nivel	FDD
Entregables a cliente	Máximo 2 semanas	*De* 2 a 4 semanas	*FDD*
Disposición de iteraciones	Se agrupan requerimientos por *Feature Set*	Se congelan los requerimientos por *Sprint*	FDD
Orden de tareas	Orden estricto	Orden flexible	*Scrum*
Capacidad de adaptación	Difícil de adaptar	Fácil de adaptar	*Scrum*
Detección de fallas	Fácil detectar fallas	Difícil detectar fallas	FDD

Tabla 4-1: comparación entre *Feature Driven Development* y Scrum

La tabla 4-1 muestra en la columna GDR que enfoque se aplicaba más al proyecto. Habiendo analizado y ponderado la importancia de cada característica para el proyecto, se decidió que lo más importante era poder detectar las fallas a tiempo para tomar las acciones necesarias; y poder cumplir con entregas al cliente con mayor frecuencia proporcionaba un mayor volumen de *feedback*. Este *feedback* se consideró como de suma importancia para reducir la incertidumbre de algunos requerimientos planteados.

Por lo tanto la estructura del proyecto dada por la definición de las actividades y la evolución del producto está dado por FDD.

4.3 Modelos de procesos

Al comienzo del proyecto, el equipo definió los principales procesos que se llevarían adelante a lo largo del mismo. Esto se hizo con la finalidad de poder ver en todo momento los pasos a dar a medida que el proyecto avanza. Los procesos claves a definir fueron el de requerimientos, el de construcción y el de calidad.

4.3.1 Proceso de requerimientos

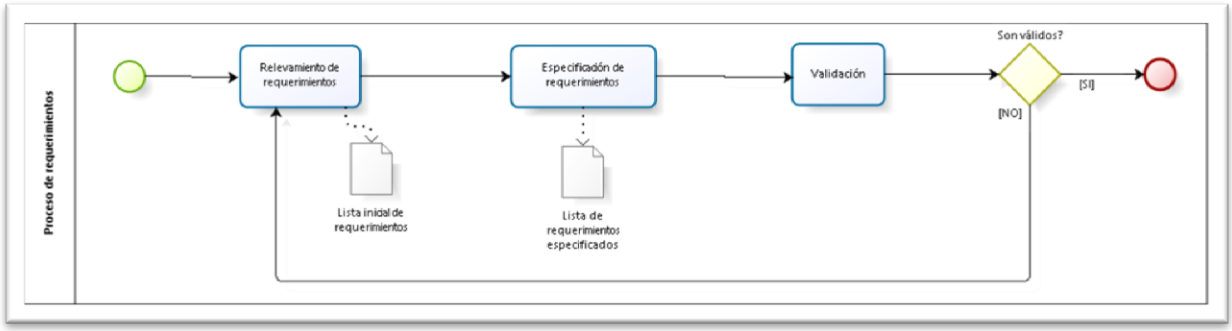

Figura 4-2: proceso de requerimientos

El proceso de requerimientos consistió en una etapa de relevamiento en que se discutió con el cliente que funcionalidades esperaba del producto, a la vez que se le presentó nuestra propuesta de funcionalidades. Luego de definido que quiere el cliente, se especificaron los *features* y se le presentó la lista obtenida al cliente para obtener la validación. En caso de que el cliente no validara la lista, se volvía a realizar el proceso.

4.3.2 Proceso de construcción

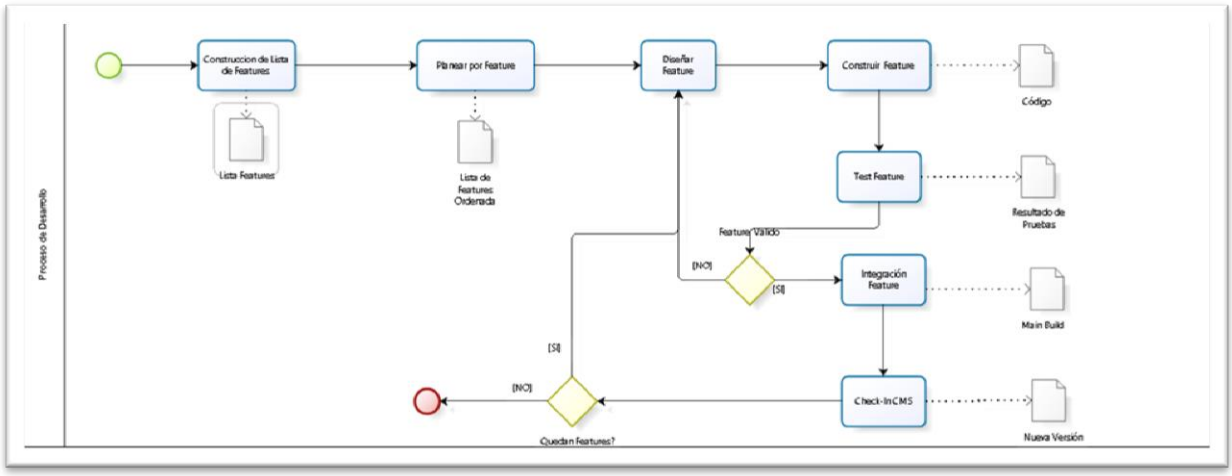

Figura 4-3: proceso de construcción

El proceso de construcción comienza en el momento que finaliza el proceso de requerimientos. Una vez obtenida la lista de *features* validada por el cliente se procede a planificar por *feature*, donde se ponderan los *features* en función de su relevancia para el proyecto. Luego, para cada conjunto de *features* a implementar en cada iteración, se diseñó, se construyó y se probó el *feature*. Si en las pruebas se obtuvieron resultados favorables, se integra a lo hecho y se sube al repositorio.

4.3.3 Proceso de calidad

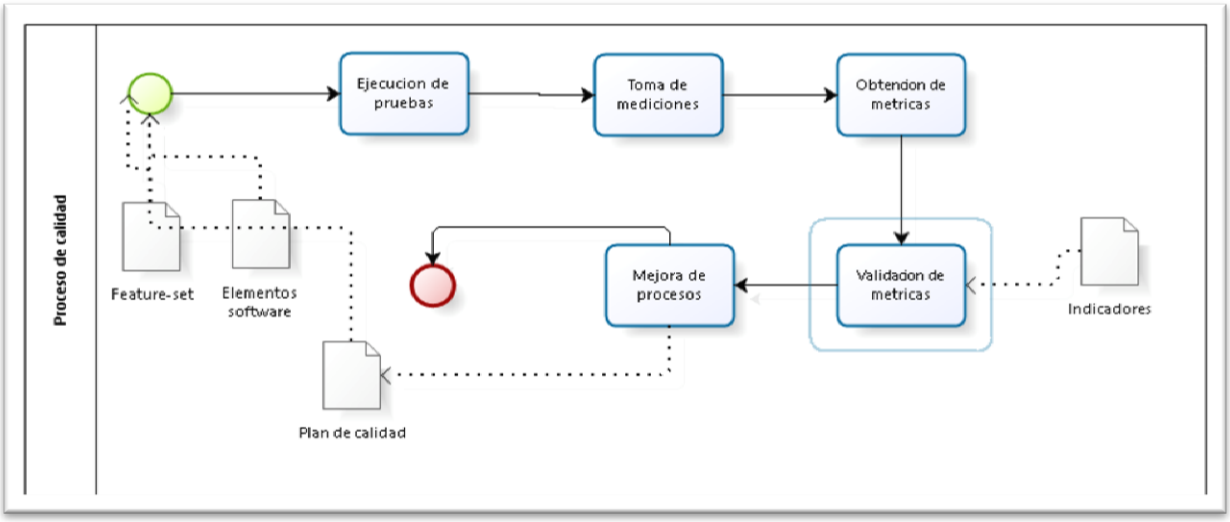

Figura 4-4: proceso de calidad

El proceso de calidad se ejecutó transversalmente al de requerimientos y construcción. Se ejecutaron pruebas y se tomaron mediciones, para luego obtener métricas en base a la información recolectada. Luego, considerando los indicadores definidos al comienzo del proyecto, se validaron las métricas y se mejoró el proceso.

5 Arquitectura de *software*

El presente capitulo describe la arquitectura del sistema. Comienza con ciertos conceptos que se manejan a lo largo del capítulo y que son necesarios para comprender las decisiones que el equipo tomó. Explica los atributos de calidad deseados en la arquitectura y las decisiones que se tomaron para alcanzarlos. Por último se presentan diversos diagramas que explican gráficamente la arquitectura completa.

5.1 Atributos de calidad

A continuación se detallan los atributos de calidad que el equipo consideró al momento de crear la arquitectura. Se presentan en orden de relevancia, de mayor a menor, por lo que los atributos más relevantes y que más se tomaron en cuenta para crear la arquitectura fueron el rendimiento y la usabilidad.

Rendimiento

Ya que existen otros productos en el mercado que satisfacen necesidades similares, es esencial que el producto cumpla con los tiempos de respuesta que la competencia cumple. Estos tiempos de respuesta luego de consultar con usuarios finales son percibidos como rápidos. Por lo tanto este es uno de los atributos más importantes de este sistema. También dado que el sistema será usado en un ambiente empresarial se debe poder tener la mayor cantidad de usuarios trabajando en paralelo.

Se fijó como cota que la generación del reporte debe darse en menos de 1 minuto y que el sistema soporte hasta 20 usuarios en simultáneo trabajando sin aumentar los tiempos de respuesta de la aplicación.

Usabilidad

Ya que los sistemas de la competencia son percibidos por los usuarios finales como fáciles de usar, es sumamente importante que el sistema desarrollado también lo sea. De lo contrario este no podrá ser aceptado ya que no tendrá un buen posicionamiento en el mercado que lo utilizará. Por lo tanto es otro de los atributos clave del sistema.

Se desea que el usuario pueda generar diversos reportes de una forma sencilla y práctica, para esto se buscó generar un sistema de fácil aprendizaje para el usuario en cuanto a la generación de reportes y principalmente el acceso a la información deseada. Se adecuará a las expectativas del usuario a medida este las comunique u obtengamos información que implique complejidad de uso. Se propuso crear un sistema que tan solo con una capacitación de 30 minutos el usuario será capaz de percibirlo como intuitivo. Para que inicialmente sea intuitivo, se determinó que siempre deberá ser visible de forma clara la tarea que se esté realizando, los textos y botones deben ser claros y explicativos, y cualquier control deberá ser auto explicativo.

Extensibilidad

El usuario no deberá ver restringida la creación de reportes por la lógica definida ni por la estructura de los componentes del sistema, deberá tener acceso a todos los atributos y poder generar cualquier reporte con ellos. Esto es un gran factor de diferenciación a los productos existentes, los cuales implican en su mayoría horas de desarrollo y personalización.

En concreto mediante una interfaz definida por el equipo el usuario podrá obtener nuevas entidades o datos, sin necesidad de realizar modificaciones al código existente. Además se cuenta con un mecanismo de conversión de datos entre los datos de origen y los datos que ve el usuario, el cual puede modificarse en tiempo de ejecución.

Seguridad

Se desea contar con un log de auditoría para poder auditar las operaciones realizadas por los usuarios en un futuro.

El sistema al ser pensado para funcionar en un entorno de intranet corporativa no se definió como necesario incorporar mayores aspectos de seguridad. Se depende de una buena seguridad de la red empresarial.

Modificabilidad

Dado que la base de datos del ERP puede sufrir modificaciones es necesario contar con algún mecanismo de modificabilidad para que el impacto de estos cambios sea mínimo.

5.2 Factores clave del diseño de *software*

5.2.1 Concurrencia

Partiendo de lo visto en los capítulos "Arquitectura" y "Requerimientos" surge la necesidad de manejar múltiples usuarios en simultaneo, lo cual se deriva en un estudio del uso de la concurrencia. Para poder disminuir la carga del sistema se incluyo la tecnología Quartz para manejar con hilos de forma asíncrona todas las tareas que no fueran esenciales para el usuario

Para la planificación de reportes se implementó un planificador, en el cual el usuario guarda una planificación en la base de datos y luego gracias a la tecnología de Quartz podemos lanzar un hilo que se encargue del proceso de generar y enviar por correo un reporte.

Quartz es un proyecto de terceros que encapsula la lógica necesaria para trabajar con múltiples hilos de una forma simple y ágil. Entre otras cosas permite al programador enfocarse en como diseñar un sistema multi-hilos fácil de mantener en lugar de utilizar horas en sincronizar y asegurar que todos los hilos funcionen.

5.2.2 Persistencia de los datos

La persistencia de datos fue un aspecto clave en el proyecto y al que se le dedicó mucho tiempo. Por un lado, para almacenar los datos del usuario e información necesaria para el funcionamiento del sistema, se utilizó una base de datos relacional MySql. Ésta base de datos permite una fácil integración con Grails y es una de las tecnologías de base de datos más populares, por lo cual existe abundante documentación acerca de ella y está probada extensivamente.

Por otro lado, se utilizó una base de datos orientada a columnas llamada Infobright. Ésta base de datos está hecha para optimizar los tiempos de selección y es la que permite manejar grandes volúmenes de datos en fracción de segundo. En ella se almacena toda la información que el sistema recibe desde PSIG ERP, y es contra la cual se hacen las consultas para obtener los datos para el reporte.

5.2.3 Distribución de los componentes

Ya que inicialmente se presentó como un gran desafío cumplir con los altos estándares de performance que el cliente esperaba, y existía mucha incertidumbre en cómo lograr traducir

las consultas de lenguaje usuario a SQL, se pensó en una arquitectura orientada a componentes físicos que pudieran ser sustituidos en caso de ser necesario, con el menor impacto posible.

5.2.4 Seguridad

Ya que el sistema funcionaría principalmente dentro de una red empresarial (en varios casos solamente con conexión por VPN), la seguridad no fue un atributo de calidad esencial para el sistema. Igualmente se plantearon mecanismos de control de la seguridad ya que para el equipo resultó importante transmitir tranquilidad al cliente (por ejemplo poder realizar auditorías de acceso en caso de ser necesario). Y ya que el sistema consta con un manejo de información esencial para las decisiones empresariales se debió diseñar algún mecanismo de seguridad. En concreto se optó por guardar registros de las operaciones para saber quién hizo qué.

5.3 Decisiones de arquitectura

Ya presentados los requerimientos no funcionales y los atributos de calidad pasaremos a detallar las decisiones tomadas a más bajo nivel. El informe completo en el anexo Arquitectura del software.

5.3.1 Tecnologías

En cuanto a las tecnologías a utilizar para el desarrollo se decidió utilizar Groovy on Grails como principal *framework* de desarrollo, ya que está basado en la estructura modelo vista controlador, en la cual se basan la mayoría de los desarrollos web actualmente. Pero también se decidió utilizar PHP para algunos componentes en concreto ya que brindaba un conjunto de beneficios. Dentro de estos beneficios se encuentra la posibilidad de generar componentes muy pequeños, y por lo tanto se disminuye la probabilidad de incorporar defectos al tener que escribir menos código, posee un proceso de *deployment* más sencillo y realizar cambios o correcciones en productivo es más sencillo ya que es un lenguaje interpretado.

Las bases de datos en columnas son bases que tienen la particularidad de organizar sus datos en segmentos de columnas en lugar de filas, como lo hacen usualmente las bases de datos relacionales vistas a lo largo de la carrera. La justificación es que las bases de datos orientadas a columnas fueron diseñadas para ser muy performantes al momento de hacer la

selección de datos. No sucede lo mismo con la inserción de datos, la cual consume más tiempo que en las bases de datos relacionales.

5.3.2 Restricciones de costos y performance

Debido a la necesidad del mercado y del cliente de realizar un sistema sumamente performante, el equipo realizó una prueba de concepto, las cuales están explicadas en el capítulo gestión del proyecto. De esta POC el equipo concluyo que la mejor opción para manejar grandes volúmenes de información en un corto periodo de tiempo eran las bases de datos en columnas.

En concreto se decidió implementar MonetDB ya que era la que presentaba mejores tiempos y a simple vista no presentaba grandes complicaciones para integrarse con un lenguaje desarrollado sobre Java (Grails). Como segunda opción se manejó Infobright, una rama de MySql con la misma interfaz pero siendo una base de datos en columnas, no en filas como lo es MySql.

Se desarrolla más sobre el tema en el capítulo de construcción.

5.3.3 Evolución de la API

A AT&G Informática se le expuso una interfaz, por la cual ellos enviarían la información de sus transacciones de compras y ventas (para más detalle de cómo fue construida ver el capítulo Construcción del *Software*). Dicha interfaz fue acordada inicialmente entre el equipo y el equipo de desarrollo de AT&G y luego sufrió cambios menores bajo pedido del cliente.

La estructura de JSON, al no depender del orden y tampoco tener que definir tipos de datos, es bastante flexible. Posee los tipos de datos básicos de cualquier lenguaje de programación y para el manejo de tipos complejos u objetos se define un JSON interno. Para más detalle de cómo construir un JSON ver la referencia [16].

Al momento de construir la API se previó que podían llegar a ocurrir cambios durante el desarrollo o luego debido a alguna particularidad de algún cliente, por lo tanto se decidió utilizar una estructura de JSON para disminuir el impacto de los cambios

5.4 Diseño de interfaz de usuario

5.4.1 Principios generales del diseño de UI

Ingeniería inversa: Proceso de obtención de información o diseño a través de un producto. [17]

Modelos conceptuales: Modelo poco sofisticado y por lo tanto más simple de comprender. [18]

5.4.2 Problemas de diseño de UI

Desarrollar una UI simple de utilizar e intuitiva, pero que a su vez expusiera un gran conjunto de operaciones a realizar se planteó como uno de los mayores desafíos en lo que a usabilidad respecta.

5.4.3 Diseño de interfaces de usuario

Primero se realizaron bocetos con herramientas de maquetación de interfaz de usuario para validar con el cliente.

La base para la confección de estos bocetos fue la aplicación de ingeniería inversa basada en la observación de sistemas similares del mercado de las herramientas de *reporting*, observación de sistemas de gestión integral y por ultimo una cuota de innovación por parte del equipo de desarrollo.

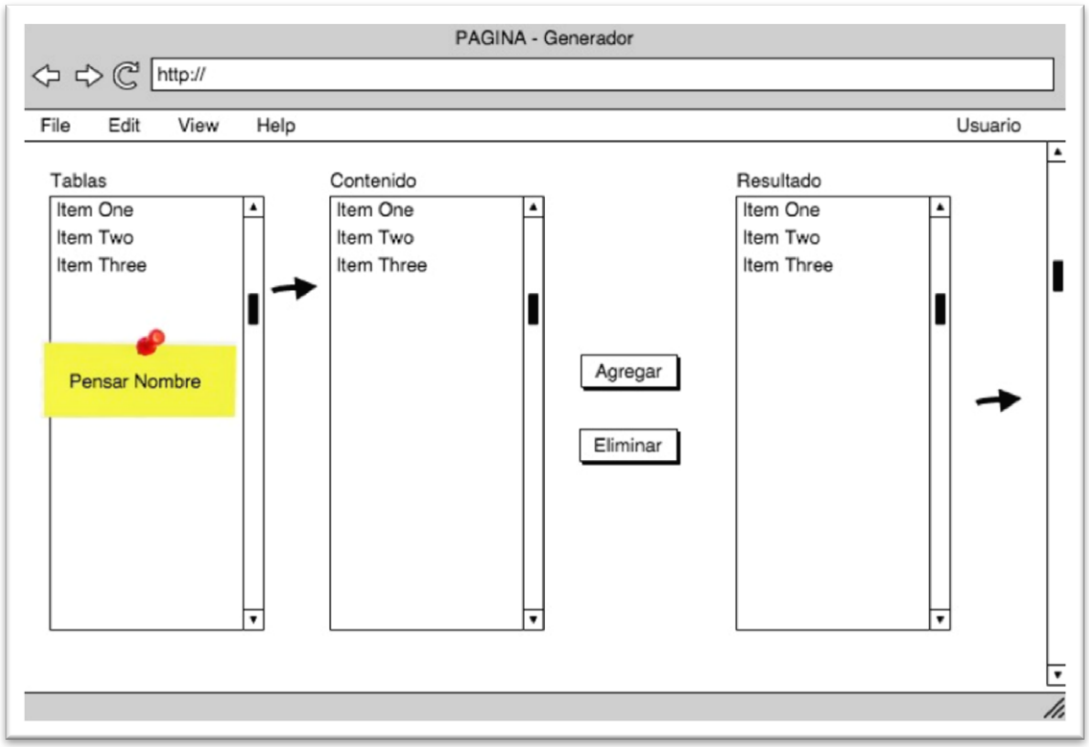

Figura 5-1: primera versión del primer paso para crear el reporte

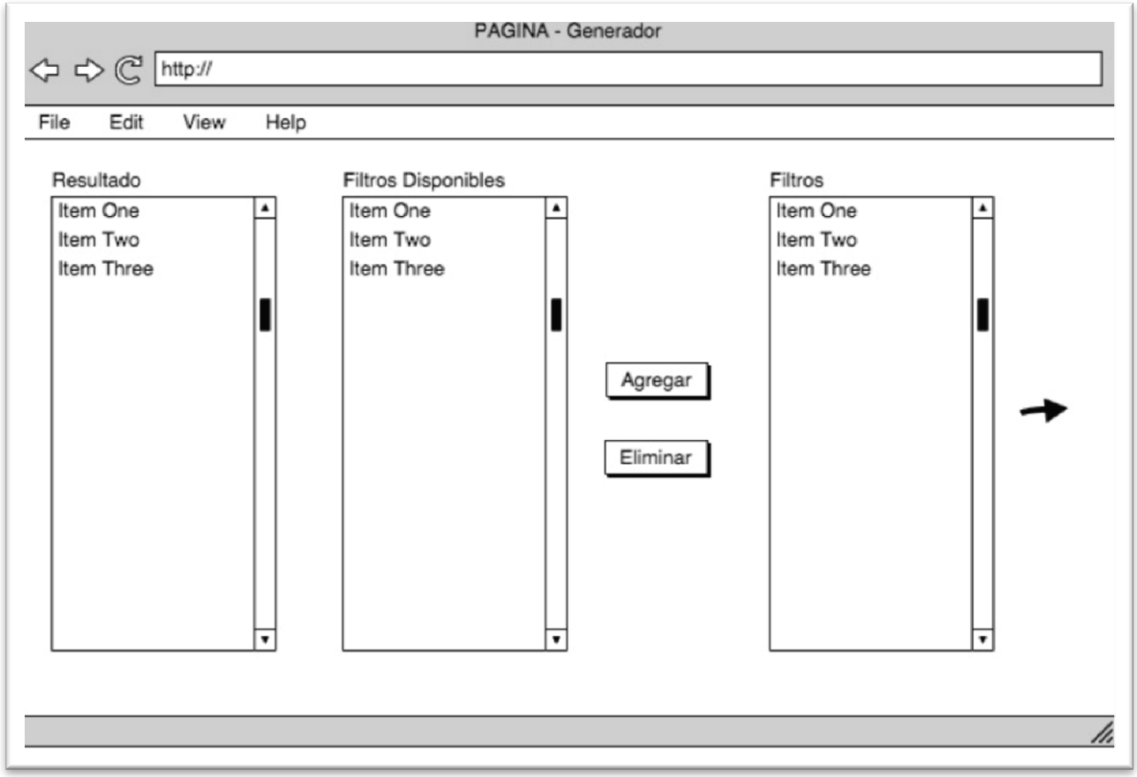

Figura 5-2: primera versión del segundo paso para crear el reporte

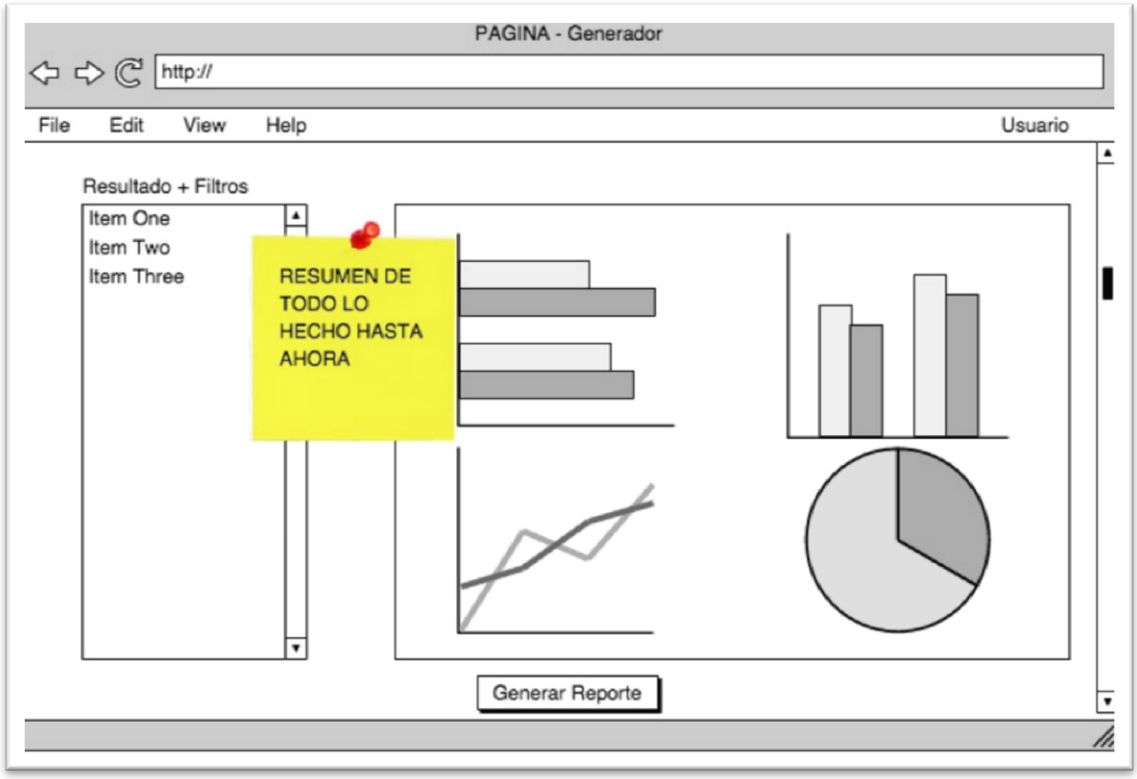

Figura 5-3: primera versión del tercer paso para crear un reporte

Una vez validados estos se procedió a realizar un prototipo funcional como primer hito del desarrollo. Este consistió en un proceso de generación de reportes por pasos, donde cada paso se enfocaba en un aspecto distinto de la información a consultar

En el primer paso se le solicita al usuario la información que en última instancia verá en el reporte. Es decir que solo debe seleccionar lo que desea visualizar, ni filtros ni condiciones.

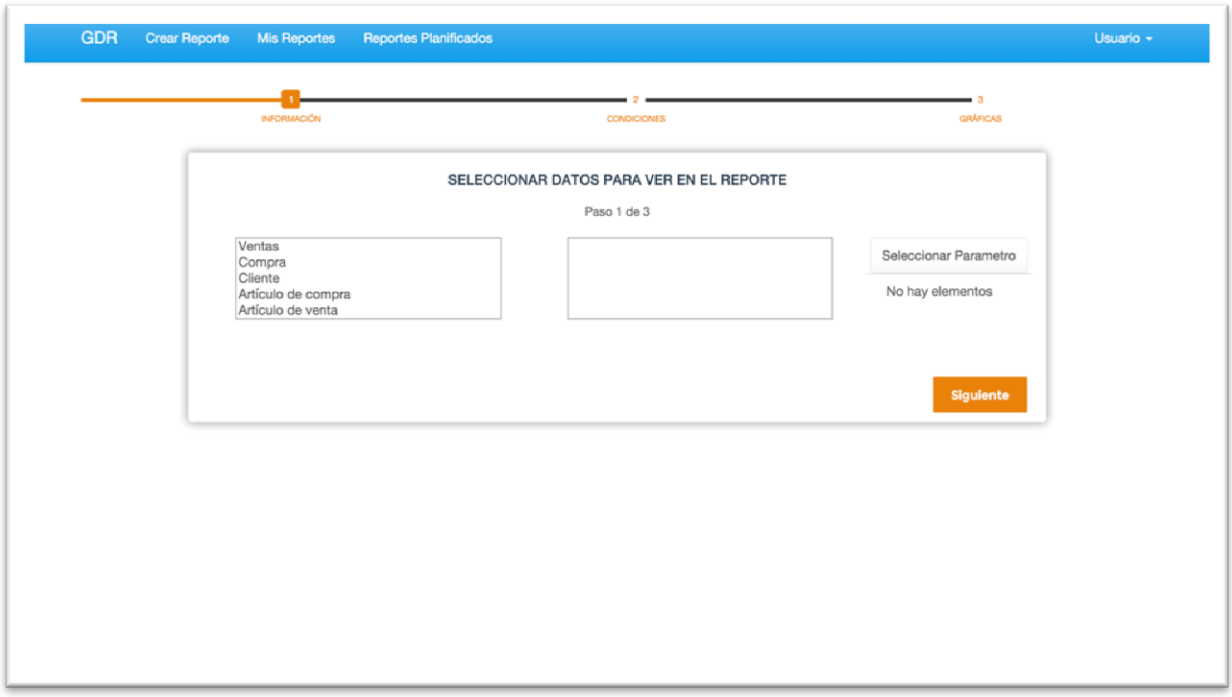

Figura 5-4: primer paso del prototipo funcional.

En el segundo paso es donde el usuario indica qué filtros o condiciones quiere aplicarle a la información que desea visualizar. Por ejemplo si quiere consultar solo la información del último mes.

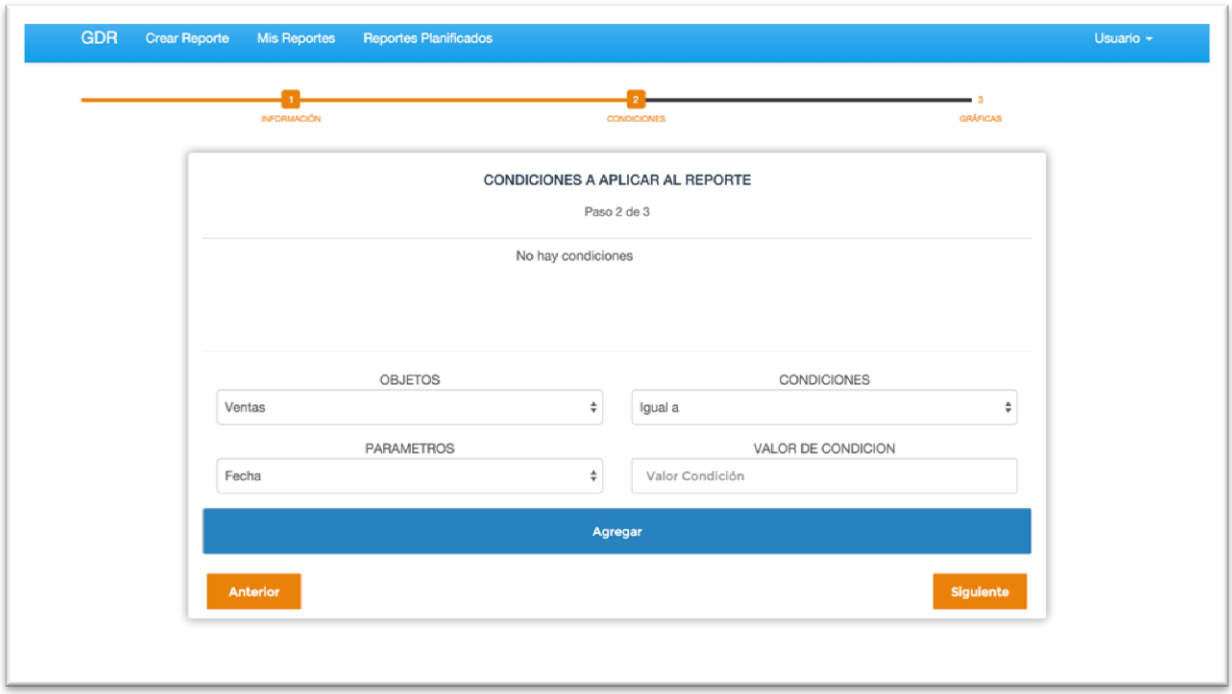

Figura 5-5: segundo paso del prototipo funcional.

En el tercer paso se debe seleccionar como desea visualizar la información que se seleccionó en la primera parte. Además debe seleccionar bajo qué criterio desea agrupar y ordenar la información.

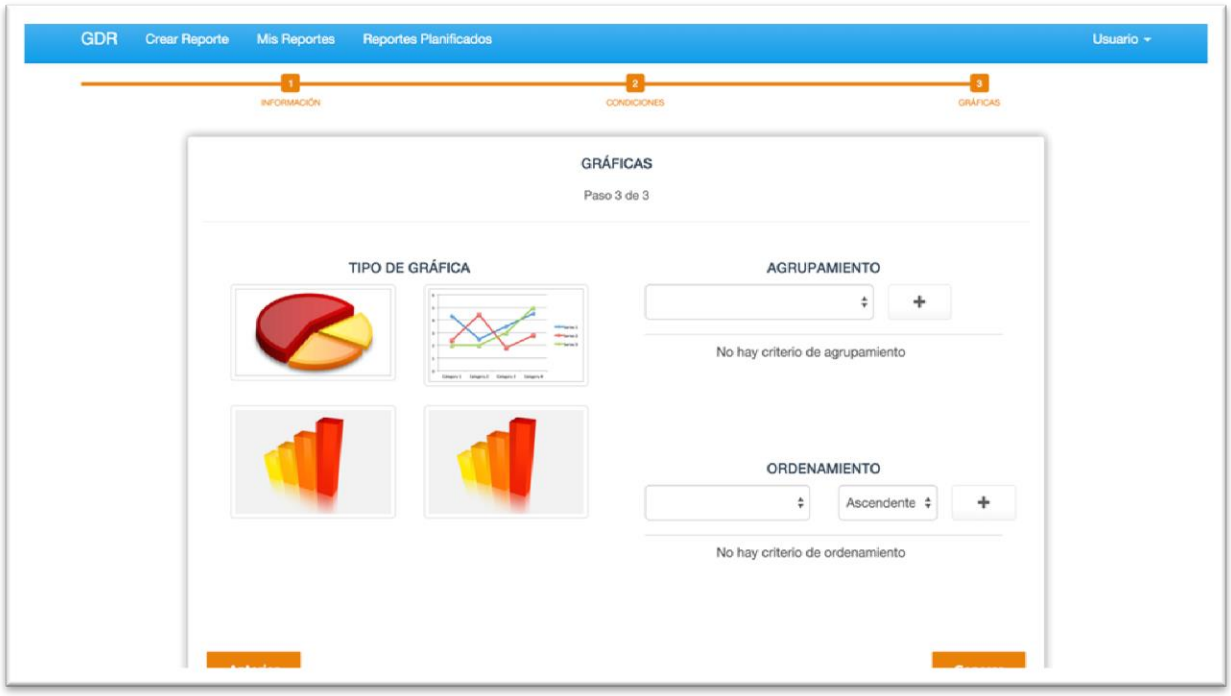

Figura 5-6: tercer paso del prototipo funcional.

A medida que el proyecto fue avanzando se implementaron pruebas con posibles usuarios finales con el fin de evaluar la usabilidad. Con estas pruebas de usabilidad buscó encontrar posibles oportunidades de mejora que hagan la interfaz más sencilla para el usuario. Ver el anexo pruebas de usabilidad para más detalles sobre las pruebas.

El equipo definió que dichas pruebas de usuario deberían ser realizadas a usuarios no técnicos ni informáticos, teniendo semejanza con el perfil de usuario que utilizará el sistema, y en lo posible no teniendo un vínculo cercano con los integrantes del equipo.

5.4.4 Diseño de presentación de la información

- Todas las páginas de la aplicación deben tener el mismo *look & feel*.
- El usuario debe tener siempre presente de forma clara que operación está realizando.
- El usuario debe tener siempre presente de forma clara que operaciones puede realizar.
- Los textos deben ser claros y auto explicativos.
- Los botones deben ser claros, debidamente visibles y auto explicativos.

5.4.5 Proceso de UI

Se desarrollaron bocetos iniciales los cuales fueron validados entre los miembros del equipo y se procedió a desarrollar una primera versión directamente en HTML. Una vez maquetada la misma y agregando un mínimo de comportamiento para que el cliente pudiera ver datos reales (porque así lo solicitó) se procedió a validar.

Una vez finalizado el desarrollo se pasó a validar nuevamente la UI, pero ahora siguiendo un flujo completo del proceso en el cual se veían implicados los componentes de UI.

Realizar pruebas de usabilidad con posibles usuarios finales para detectar errores u oportunidades de mejora. Si los resultados son positivos realizar nuevamente las pruebas de usabilidad ante la incursión de cambios, pero si los resultados son negativos volver a evaluar el desarrollo de la UI.

5.5 Anotaciones del diseño del *software*

5.5.1 Vistas estáticas

A continuación se presenta un resumen de las vistas estáticas de la arquitectura. Para más detalle dirigirse al anexo Arquitectura del software.

5.5.1.1 Despliegue

Existen variantes en cuanto a la disposición de los componentes en los distintos nodos, pudiendo cada componente ser ubicado en un nodo aislado. También resultaría posible ubicar toda la solución en un único nodo, pero no es la topología ideal que se plantea. El equipo creyó que con la topología planteada se puede proponer un servidor con las características mínimas necesarias para un adecuado funcionamiento de cada elemento del sistema.

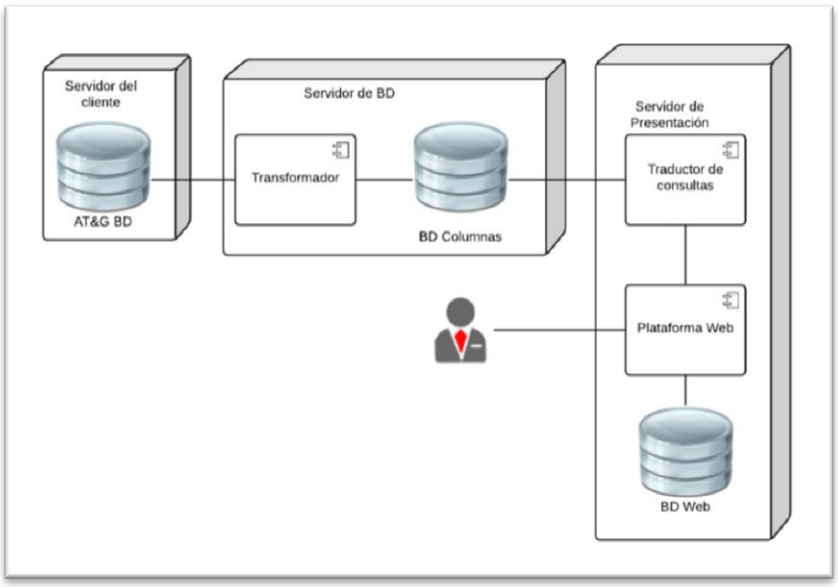

Figura 5-7: diagrama de despliegue

5.5.1.2 Componentes

Transformador

Es el componente encargado de recibir los datos del ERP y mediante un proceso de transformación insertar los datos la base de datos del sistema. Para esto, se implementó una interfaz que recibe las transacciones que envía AT&G Informática y se guardan localmente en el servidor que las recibe. Luego, mediante un *checkpoint* que permite saber cuáles son las nuevas transacciones, se insertan, actualizan o borran las transacciones, según corresponda, en la base de datos en columnas.

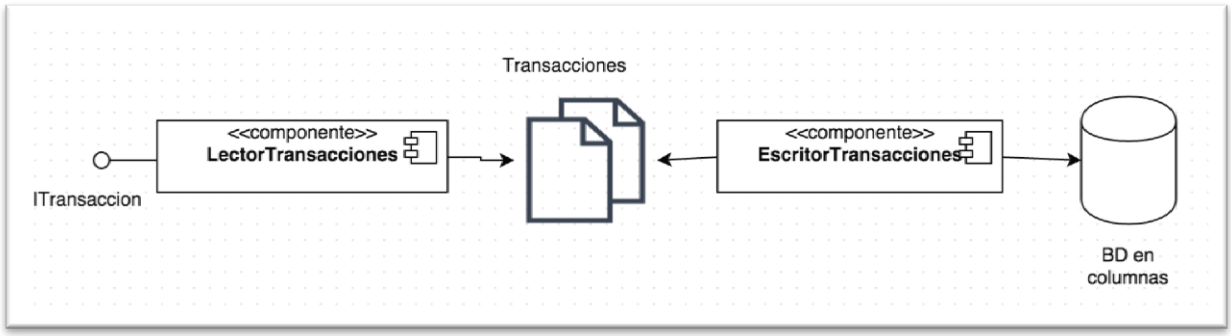

Figura 5-8: diagrama de componentes del transformador

Traductor de consulta y plataforma web

El intérprete será el encargado de traducir las consultas complejas de los usuarios para que luego el manejador de consultas pueda obtener los datos de la base de datos. Para consultas simples de obtención de datos estándares se utilizará la interfaz IInterprete.

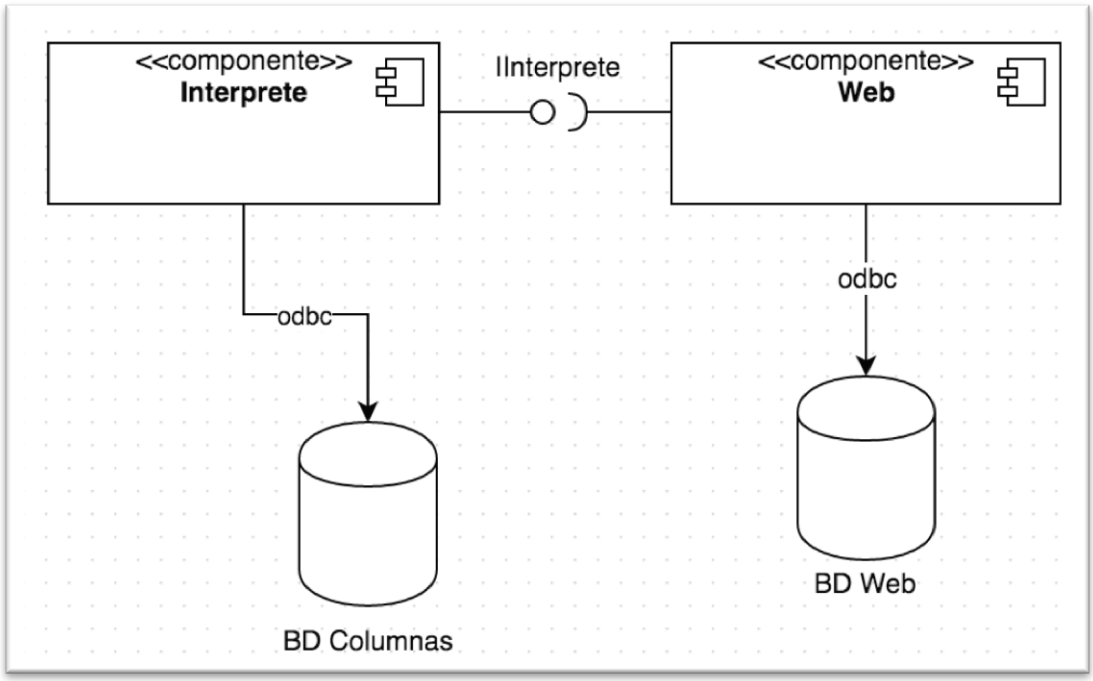

Figura 5-9: diagrama de componentes del traductor de consulta y plataforma web

5.5.1.3 Módulos

Web

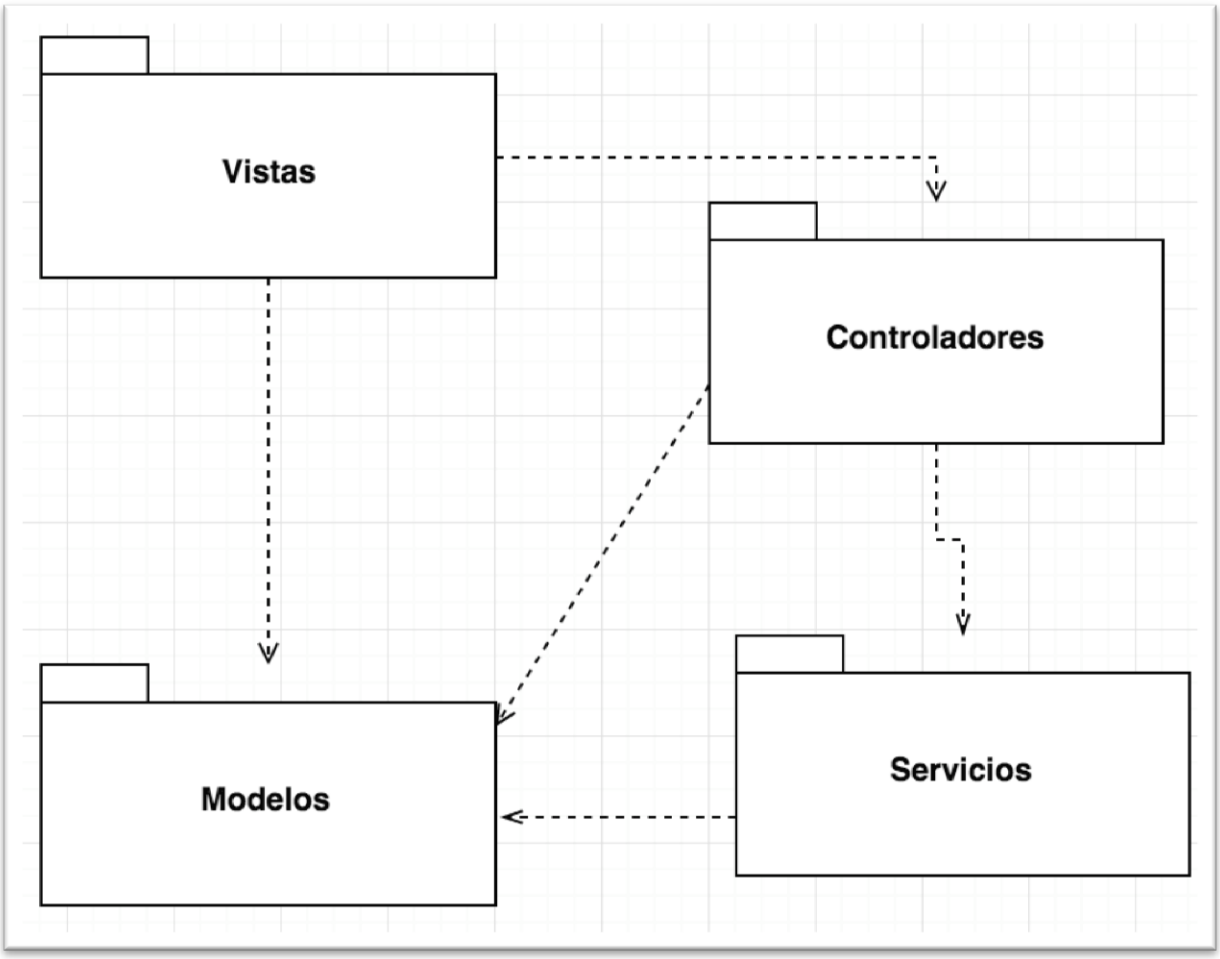

Figura 5-10: diagrama de módulos de plataforma web

Es un modelo simple ya que se buscó que fuese simple de mantener a futuro, y simple de programar en el corto plazo. Principalmente el equipo decidió utilizar una arquitectura MVC ya que era la arquitectura con la que se contaba mayor conocimiento y experiencia.

Interprete

Este componente se encarga de recibir las consultas en lenguaje usuario e interpretarlas a lenguaje SQL.

La creación de la sintaxis SQL fue encapsulada lo más posible de forma que el impacto de cambio en un futuro sea el menor posible

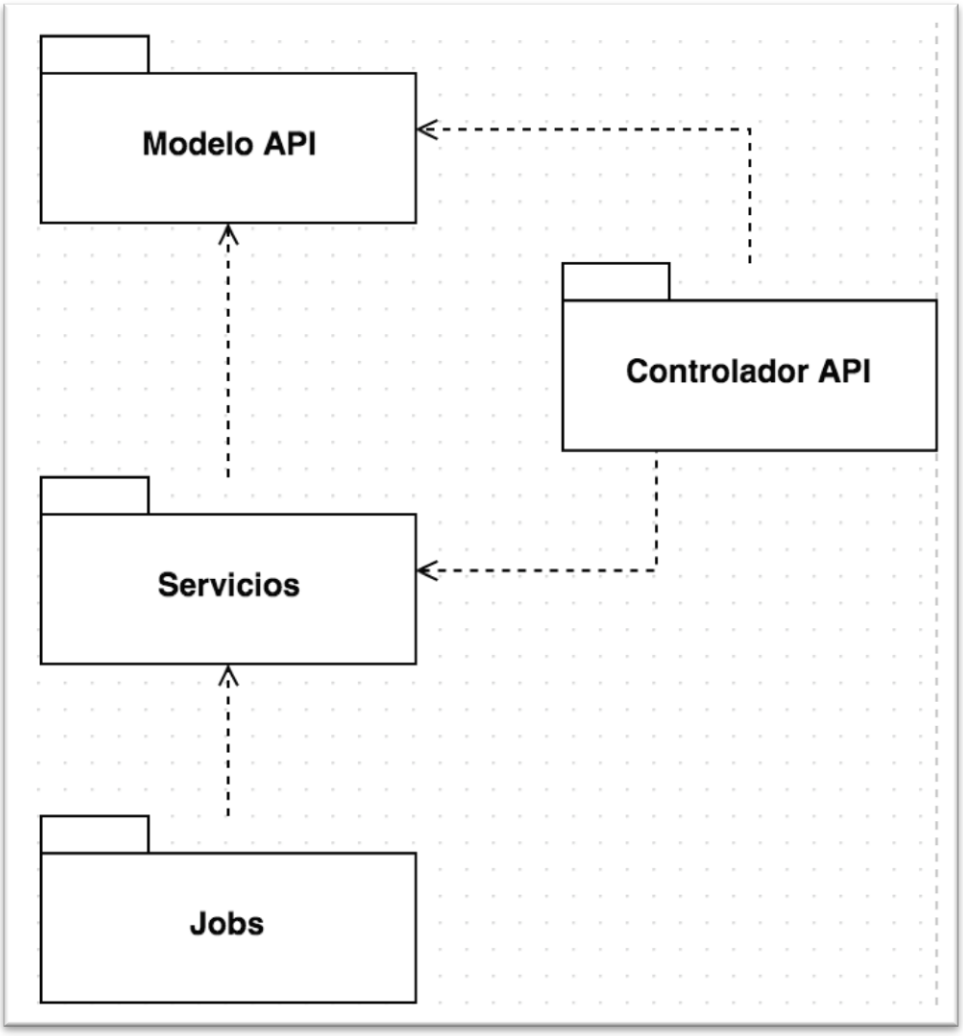

Figura 5-11: diagrama de módulos del traductor de consultas

Controlador API es quien implementa la interfaz expuesta a la Web y mediante el modelo maneja los tipos de objetos definidos en el contrato. Luego utiliza una capa de servicios para realizar la lógica necesaria para procesar los pedidos.

Y por último hay una capa de *jobs* que funciona de forma totalmente asincrónica y se utilizó para realizar eventos o tareas programadas.

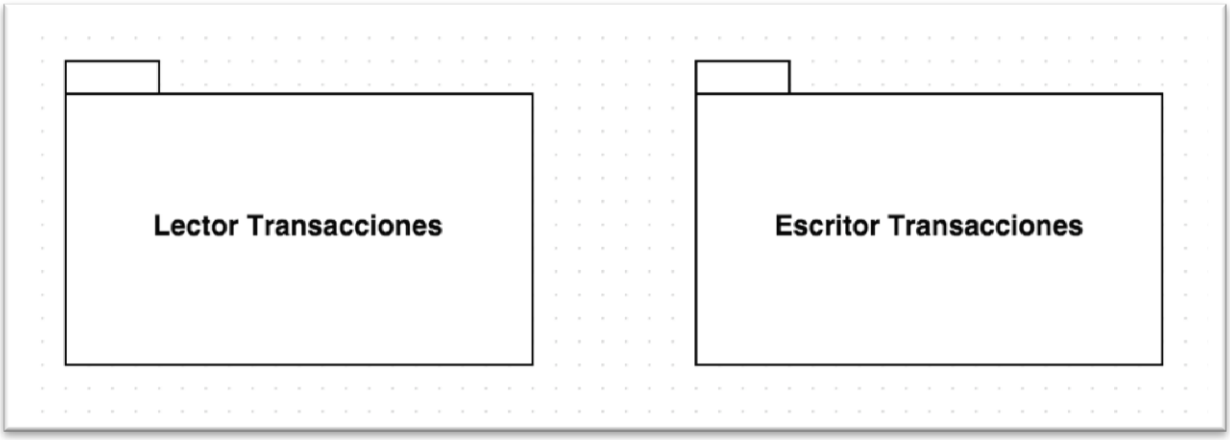

Figura 5-12: diagrama de módulos del Transformador

El anterior diagrama muestra capas que corren en el componente "Transformador" que era el encargado de recibir las transacciones del ERP, procesarlas e insértalas en la base de datos en columnas

Lector Transacciones API es la interfaz de comunicación con los servidores del cliente, los cuales nos envían toda transacción nueva, la cual nosotros recibimos y la guardamos en un directorio del servidor propio.

Luego, el Escritor Transacciones es el encargado de, cada determinado tiempo, tomar de dicho directorio todas las transacciones nuevas e insertando en la base de datos propia solo aquellas que pasan el proceso de validación de formato.

5.6 Tecnologías involucradas en la arquitectura

Las herramientas de diseño de *software* utilizadas fueron:

- UML 2.0

6 Gestión del Proyecto

En este capítulo se evidencia y describe la gestión que se llevó a cabo durante el proyecto. El equipo considera fundamental la gestión precisa en todos los ámbitos del proyecto para lograr los objetivos propuestos. Se podría dividir la gestión en tres etapas: investigación, desarrollo y producción.

6.1 Investigación

La primer etapa, investigación, corresponde a todo aquello realizado previo al comienzo del desarrollo, la siguiente etapa, de forma de adquirir conocimientos que sean útiles en la realización del proyecto y la toma de decisiones.

Se centra la investigación en tecnologías desconocidas, o no tan conocidas, para los integrantes, debiendo adquirir conocimientos de las mismas para su aplicación tanto directa como indirecta en el proyecto. Sobre estas áreas se busca comprender las bases de las mismas, herramientas utilizadas en dichas áreas y cuáles son las mejores opciones que se adaptan a las necesidades de nuestro proyecto sobre dichas áreas.

Se puso especial interés en los siguientes puntos:

- Bases de datos orientadas a columnas: la base de datos es un aspecto crítico en nuestro proyecto, puede determinar tanto el éxito como el fracaso del sistema, por lo que elegir la tecnología correcta era crucial. Desde el comienzo del proyecto el equipo sabía de la tecnología de base de datos en columnas y de sus beneficios, pero para tomar una decisión había que realizar una investigación empírica de forma tal de entender cuál de las tecnologías disponibles podía ser útil para el proyecto.

- Tecnología Groovy on Grails: existen *frameworks* que permiten generar aplicaciones web con mayor facilidad que otros. Hay muchas opciones posibles, pero la correcta elección llevaría a que el equipo encontrara más o menos complicaciones tecnológicas a lo largo del proyecto.

- *Reporting*: se investigaron algunas de las herramientas de *reporting* del mercado en primera instancia para evaluar la viabilidad de la idea planteada y ver que funciones ofrecían dichas herramientas. Luego el equipo se interesó en analizar la interfaz de usuarios, cuáles eran sus fortalezas y debilidades.

6.1.1 Bases de datos en columnas

Las bases de datos en columnas son bases que tienen la particularidad de organizar sus datos en segmentos de columnas en lugar de filas, como lo hacen usualmente las bases de datos relacionales vistas a lo largo de la carrera. La justificación es que las bases de datos orientadas a columnas fueron diseñadas para ser muy performantes al momento de hacer la selección de datos. No sucede lo mismo con la inserción de datos, la cual consume más tiempo que en las bases de datos relacionales.

Se llegó a la conclusión que las mismas serían aplicables al desarrollo del proyecto, con vistas en la disminución de los tiempos de respuesta a consultas que abarquen un caudal de datos importante, viendo que este caso podría darse con frecuencia en ciertos clientes. Dentro de las posibles tecnologías se optó por utilizar *MonetDB* la cual ofrece menores tiempos de inserción, con respecto a otras bases de datos en columnas, y un promedio de tiempo, en la selección de datos, inferior al resto. Para evaluar cuál era la más apropiada se realizar pruebas de desempeño únicamente, dejando de lado otros aspectos como el licenciamiento o la integración con otras tecnologías. Dicha omisión jugó un papel importante más adelante en el proyecto, que como puede verse detallado en el capítulo de construcción, se decidió cambiar a Infobright, siendo esta la segunda mejor opción en cuanto a tiempos de inserción y selección.

A continuación las gráficas que prueban por qué MonetDB fue la mejor opción:

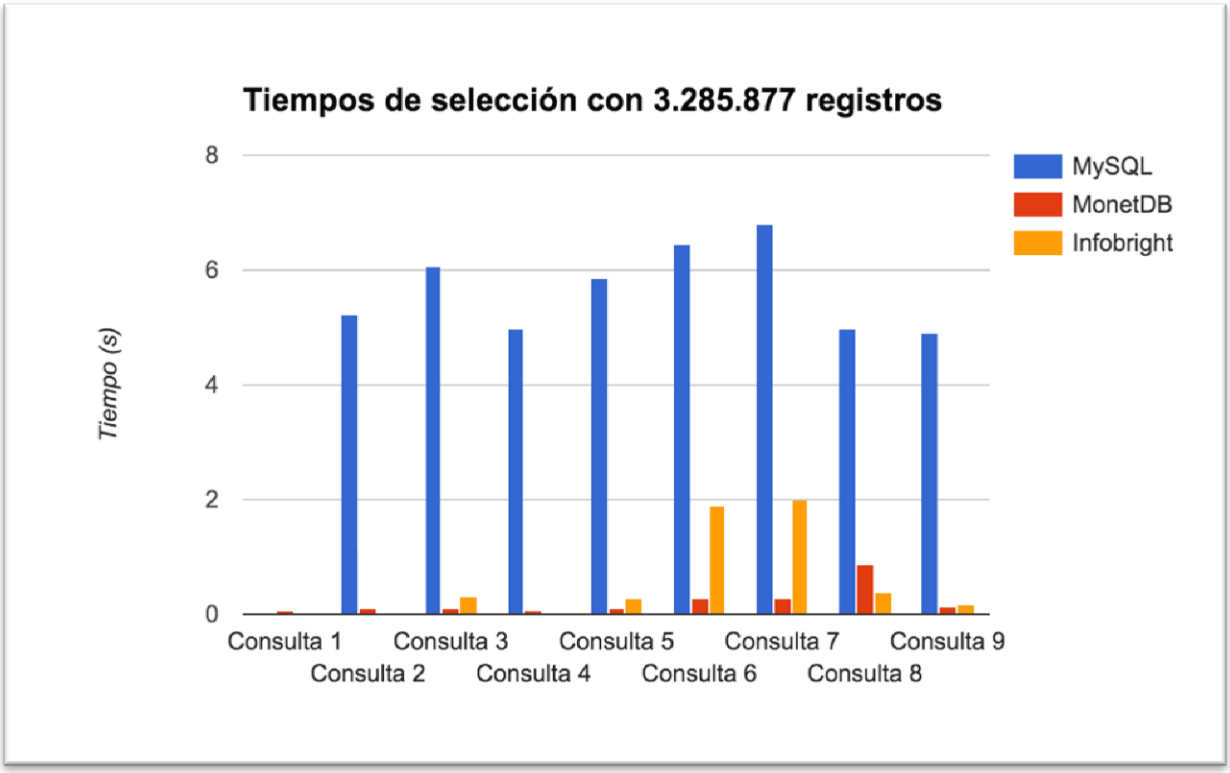

Figura 6-1: gráfica de tiempos de selección con 3.285.877 registros

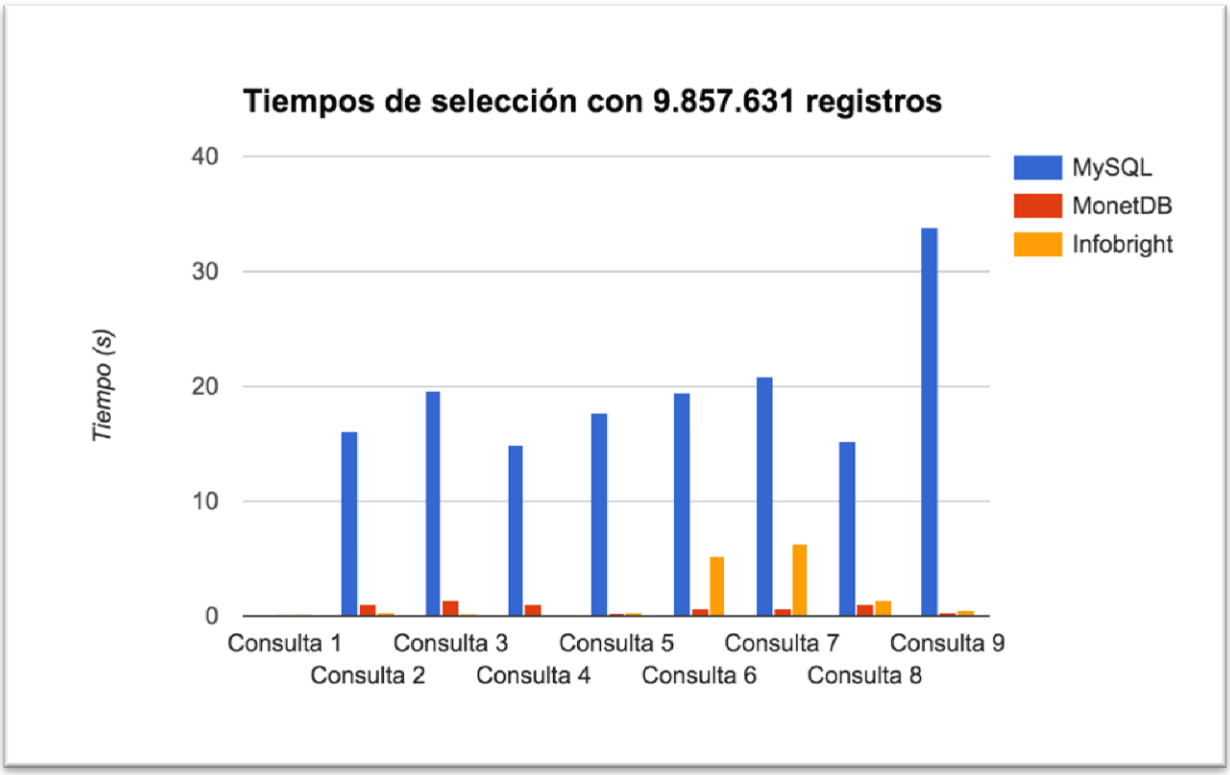

Figura 6-2: gráfica de tiempos de selección con 9.857.631 registros

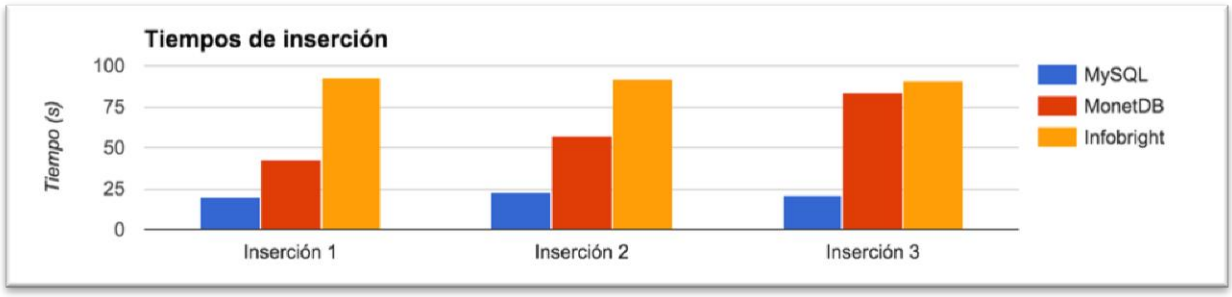

Figura 6-3: gráfica de tiempos de inserción de 3.285.877 registros

En cuanto a la inserción de datos, el equipo concluyó que la mejor estrategia sería realizar inserciones por lote. Esto se debe al bajo rendimiento de éstas tecnologías al momento de insertar nuevos registros, como se ve en la Figura 6-3. Toda la información recibida desde PSIG ERP será almacenada en un archivo de texto y una vez cada cierta cantidad de tiempo se ejecuta un proceso que insertará toda la nueva información en la base de datos para poder ser consultada desde nuestro sistema.

Para más detalle de cómo fueron realizadas estas pruebas de concepto y cuáles fueron las variantes que se analizaron dirigirse al anexo pruebas de concepto de base de datos.

6.1.2 Groovy on Grails

Grails es un *framework* de aplicaciones web, basado en el lenguaje Groovy, el cual a su vez está basado en la plataforma Java, con todas las ventajas que esto significa. Proporciona una estructura modelo vista controlador y la posibilidad de instalar funcionalidades que se encuentran en portales web que ofrecen los llamados *plugins* de Grails. Esto permite ahorrar tiempo, pudiendo conseguir e instalar funcionalidades ya hechas, que fueran necesarias en la aplicación, y que estén probadas por cientos de desarrolladores y usuarios.

El hecho que el lenguaje está basado en la plataforma Java proporciona una gran ventaja al equipo, debido al conocimiento y trabajo previo con dicha plataforma, y a la compatibilidad de dicha plataforma con los diferentes sistemas operativos. También se encontró un *driver* de MonetDB sobre la plataforma Java, lo que se supuso significaría no tener problemas en la utilización de este *framework*.

6.1.3 Reporting

Se analizaron diversas herramientas ya existentes de *reporting*, para entender un poco más su funcionamiento, que facilidades brindaban a los usuarios y cuáles eran sus enfoques. Se hizo hincapié en dos: por un lado Pentaho, un conjunto de programas que ofrecen inteligencia empresarial entre lo que ofrecen la generación de reportes e informes, y por el otro, Ideasoft O3, un producto de similares prestaciones al anterior, que ofrece la generación de reportes estáticos.

6.2 Desarrollo

La segunda etapa corresponde al desarrollo, la misma engloba todo lo relacionado a la construcción del sistema. El equipo decidió seguir los principios de las metodologías ágiles, adoptando FDD como metodología de trabajo, por las razones que se mencionan en el capítulo Metodología de trabajo. Se sigue un desarrollo iterativo incremental en espiral, que permitirá crear nuevas versiones del *software* en iteraciones de dos semanas.

Se consideró importante la elección de herramientas que ayuden y favorezcan a la buena gestión en la etapa de desarrollo. Luego de una búsqueda de alternativas, se optó por Trello, Slack, Toggl y Google Drive, herramientas explicadas en las siguientes secciones.

6.2.1 Trello

Es una aplicación web para la gestión de proyectos. La misma permite establecer un plan de actividades con responsables de las mismas, fecha límite y *checklist* de cosas que implica dicha actividad. Las actividades pueden estar divididas en diferentes conjuntos, que podrían considerarse estados de la actividad, y una actividad puede ir variando su estado. En este caso se utilizó este modelo, donde se tiene columnas para actividades a realizar, actividades que se están realizando y actividades realizadas.

A continuación una captura de pantalla de la herramienta:

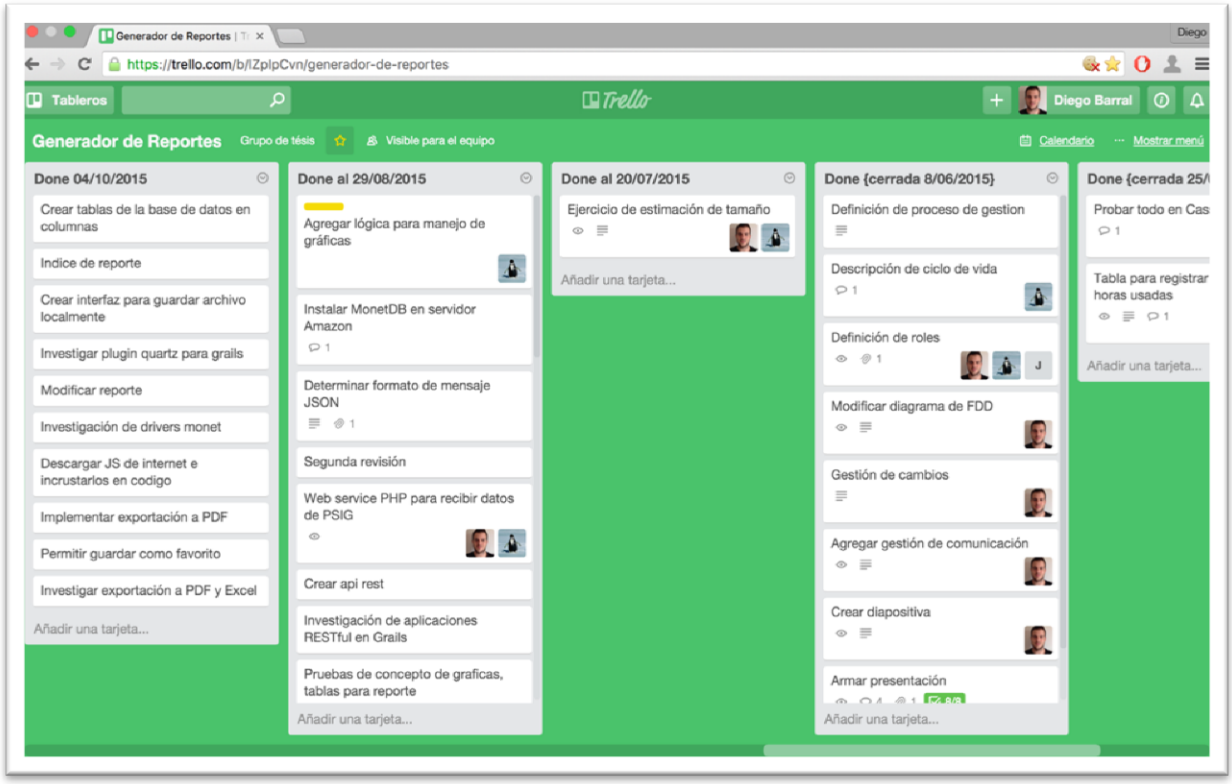

Figura 6-4: captura de pantalla de Trello

6.2.2 Slack

Es una herramienta orientada a la comunicación en equipo. La misma ofrece una estructura de *chats* ordenado y organizada, pudiendo organizar por temas, por grupos o también se puede mandar un mensaje directo. La misma ofrece aplicaciones de escritorio para los distintos sistemas operativos, así como su aplicación web.

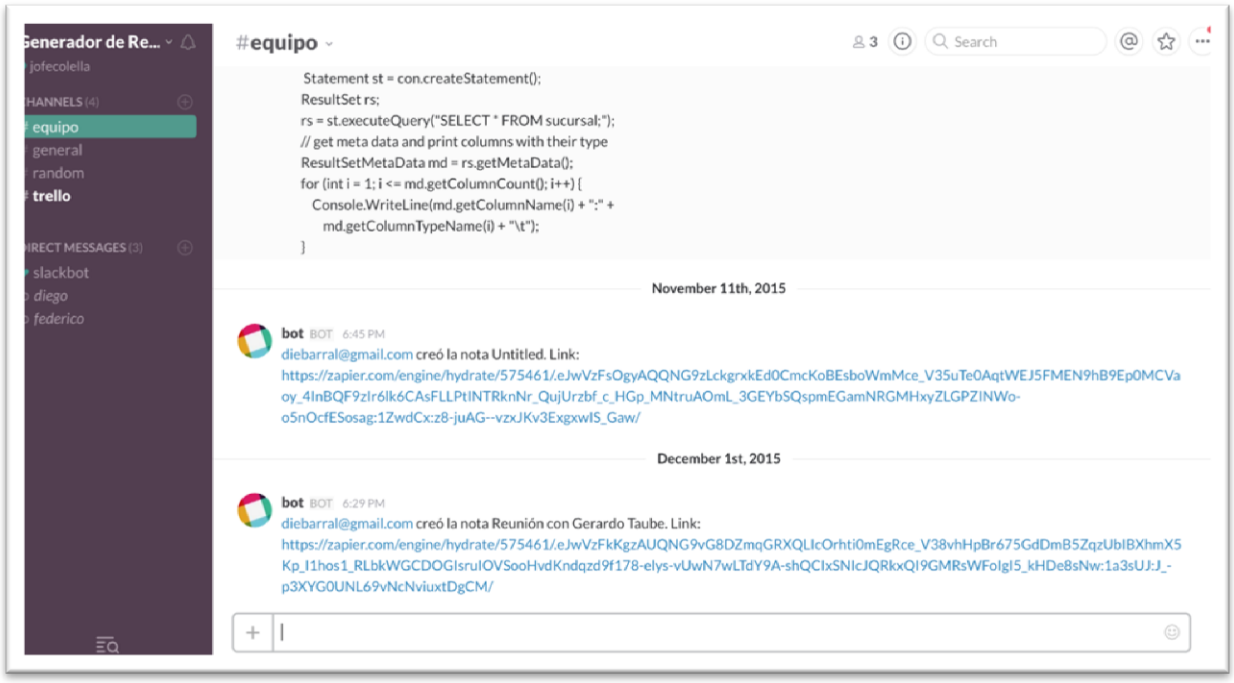

Figura 6-5: captura de pantalla de Slack

6.2.3 Toggl

Es una herramienta que se utiliza para contabilizar tiempo. La misma puede servir para tener conocimiento de cuánto tiempo consumió una cierta actividad, o mismo cuánto tiempo le dedico un integrante al proyecto en cierto intervalo de fechas. Esto permite realizar análisis de rendimiento, comparar estimaciones con valores reales de duración, entre otras cosas.

Esta herramienta tiene la ventaja de integrarse a Trello, la otra herramienta utilizada para la gestión del desarrollo. Ofrece tanto aplicaciones web y para dispositivos móviles, como integración a navegadores como Google Chrome que hacen que sea de fácil uso y accionar.

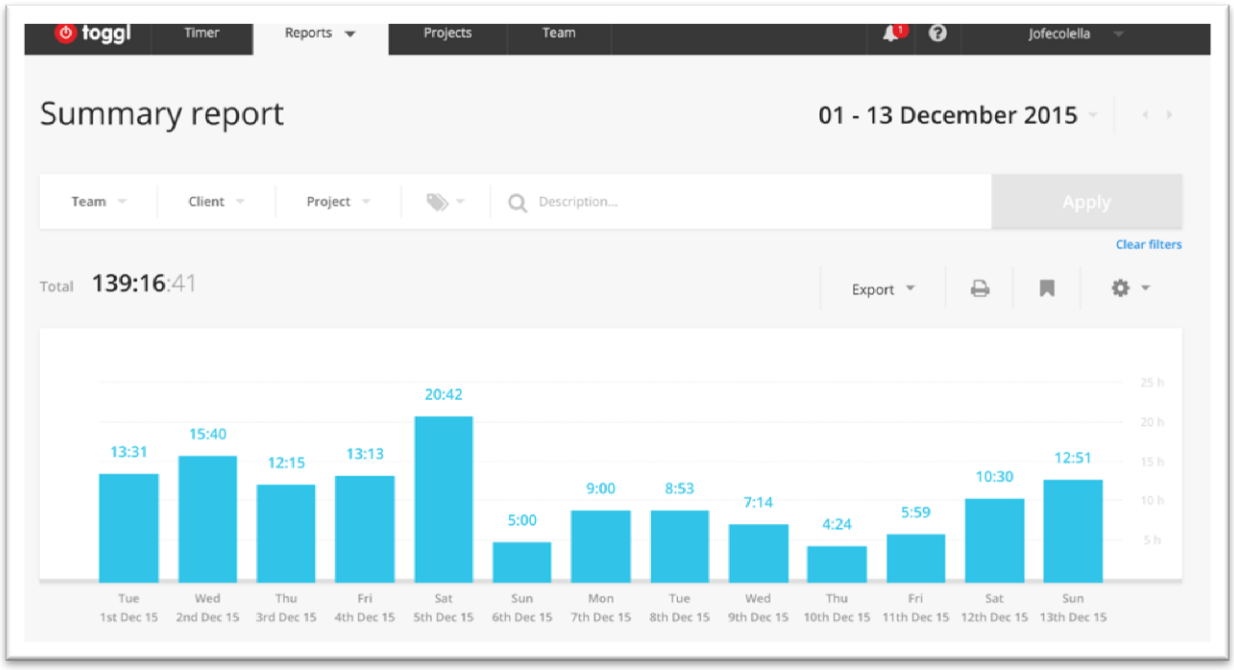

Figura 6-6: captura de pantalla de Toggl

6.2.4 Google Drive

Es un repositorio de archivos de Google que permite a múltiples usuarios acceder y trabajar simultáneamente sobre distintos documentos almacenados en una estructura de directorio. Esto permitió al equipo almacenar toda la documentación referente al proyecto de forma organizada y en un solo lugar.

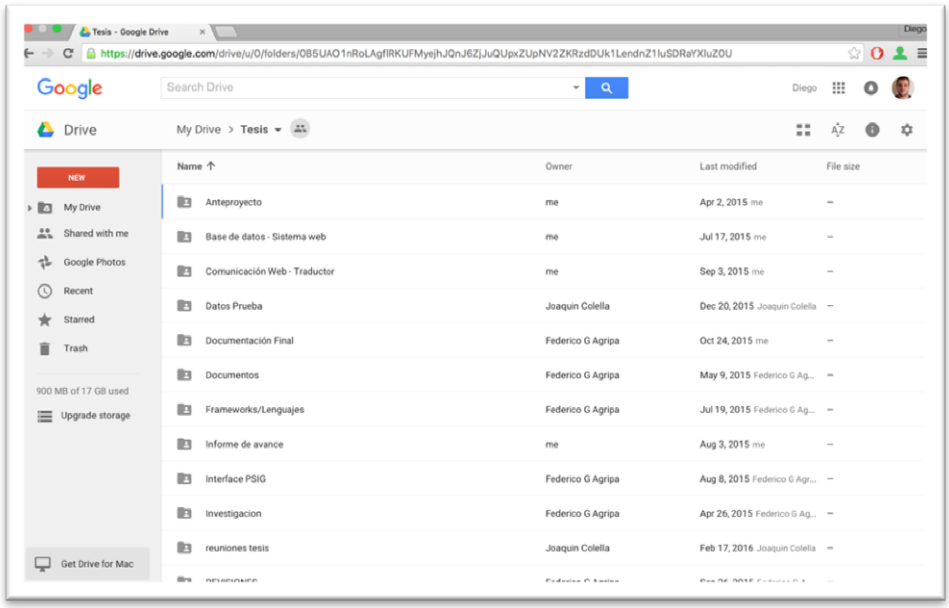

Figura 6-7: captura de pantalla de Google Drive

6.3 Producción

Por último, la etapa de producción es la única de las tres etapas que no se llevó a cabo. Esto se debió a varios factores tales como:

- Se tomó como cliente tipo uno sumamente exigente, lo cual fue un tanto ambicioso.
- La no recepción de datos a través de la interfaz planteada incurrió en un aumento de horas de pruebas y generación de datos de prueba para el equipo.
- Un cambio de alcance inesperado en los últimos meses del proyecto

Para la misma se realiza una planificación que indica cómo será la puesta en producción, la se detalla a continuación.

6.3.1 Plan de puesta en producción

Una vez se cuente con un cliente interesado en adquirir el producto, o AT&G Informática provea uno de su cartera al equipo, se debe realizar una breve demo de no más de 30 minutos para demostrar todas las características y fortalezas del producto.

Una vez el cliente acepte adquirir el producto solo queda seguir un breve manual de instalación que se detalla a continuación.

Manual de instalación

1. Instalar y configurar Infobright.
2. Instalar y configurar MySql.
3. Instalar Java.
4. Instalar servidor de aplicaciones Java (puede ser Apache Tomcat, Yeti, o similar). Se manejan varias opciones previendo variantes en cada cliente.
5. Instalar interfaz de comunicación con PSIG ERP (*deployment*).
6. Instalar traductor de consultas (*deployment*).
7. Instalar aplicación web (*deployment*).
8. Probar conectividad entre el traductor de consultas y la aplicación web mediante *requests* http.

El equipo planteó como situación ideal contar con dos servidores para el funcionamiento del sistema, tal como está planteado en el capítulo Arquitectura de *software*. Un servidor será el que almacenará la base de datos orientada a columnas y la interfaz de comunicación con PSIG ERP. El otro servidor almacenará la aplicación web y el traductor. Ambos servidores deberán contar con la instalación de Java.

6.4 Gestión de riesgos

En esta sección se presenta la identificación, seguimiento y evolución de los riesgos durante el proyecto.

6.4.1 Matriz Probabilidad - Impacto

		Impacto				
		1	2	3	4	5
Probabilidad	0,2	0,2	0,4	0,6	0,8	1,0
	0,4	0,4	0,8	1,2	1,6	2,0
	0,6	0,6	1,2	1,8	2,4	3,0
	0,8	0,8	1,6	2,4	3,2	4,0
	1,0	1,0	2,0	3,0	4,0	5,0

Tabla 6-1: matriz de probabilidad e impacto

Referencia:

	Aceptación Pasiva
	Aceptación Activa
	Mitigación
	Evitar

Tabla 6-2: tipos de acciones

A partir de la generación de esta matriz, se decidió el tipo de plan de respuesta para cada caso, producto de una discusión en la cual se terminó de definir el rango en el que se aplicaba aceptación pasiva, aceptación activa, mitigación o evitar.

En la discusión se tocó cada ítem, discutiendo que impactó el mismo generaba, cuál era la probabilidad del mismo y la ocurrencia del tiempo. Luego de calculado el valor esperado, se

vio a cuál de los grupos pertenecía y en caso de creer incorrecto el grupo al que pertenecía, se realizaba una redefinición de los rangos de cada plan. Al final, una vez terminada la discusión de cada riesgo, se repasó la decisión tomada en cada caso, para controlar que no haya ambigüedades.

Luego también, se definió que el responsable de llevar un control de los riesgos es Diego Barral, y que en caso que se deba aplicar una medida de contingencia el mismo deberá ser el encargado de informarla.

6.4.2 Lista de riesgos críticos

Numero	Descripción
1	Problemas con la conversión de base de datos del cliente
2	Problemas trabajando a distancia
3	No lograr una interfaz intuitiva
4	Contratiempos en el uso de base de datos en columnas
5	No cumplir con performance planteada

Tabla 6-3: riesgos críticos del proyecto

			Nro. de Riesgo														
			1			**2**			**3**			**4**			**5**		
It	**Fecha de inicio**	**Fecha de fin**	**IM**	**PO**	**VE**	**IM**	**PO**	**VE**	**IM**	**PO**	**VE**	**IM**	**PO**	**VE**	**IM**	**PO**	**VE**
1	20/07/2015	17/08/2015	4	0,8	**3,2**	2	0,6	**1,2**	3	0,6	**1,8**	4	0,6	**2,4**	4	0,4	**1,6**
2	18/08/2015	30/08/2015	4	0,8	**3,2**	2	0	**0**	4	0,6	**2,4**	4	0,6	**2,4**	4	0,4	**1,6**
3	31/08/2015	13/09/2015	4	0,8	**3,2**	2	0	**0**	5	0,6	**3**	4	0,6	**2,4**	4	0,4	**1,6**
4	14/09/2015	27/09/2015	5	0,8	**4**	2	0	**0**	5	0,6	**3**	4	0,6	**2,4**	5	0,4	**2**
5	28/09/2015	11/10/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0,8	**3,2**	5	0,6	**3**
6	12/10/2015	25/10/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0,8	**3,2**	5	0,6	**3**

7	26/10/2015	8/11/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	1	**4**	5	0,6	**3**
8	9/11/2015	22/11/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0	**0**	5	0,6	**3**
9	23/11/2015	06/12/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0	**0**	5	0,6	**3**
10	23/11/2015	06/12/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0	**0**	5	0,6	**3**
11	07/12/2015	20/12/2015	4	0	**0**	2	0	**0**	5	0,8	**4**	4	0	**0**	5	0,6	**3**
12	21/12/2015	03/01/2016	4	0	**0**	2	0	**0**	5	0,4	**2**	4	0	**0**	5	0,4	**2**

Tabla 6-4: evolución de los riesgos a lo largo de las iteraciones

Referencia:

It: Iteración

IM: Impacto

PO: Probabilidad de ocurrencia

VE: Valor esperado (IM x PO)

La gráfica de la evolución de los riesgos a lo largo de las iteraciones se puede ver en la figura 6-8.

La evolución de los riesgos críticos permite hacer un seguimiento del proyecto pudiéndose ver problemáticas que se presentaron, las cuales en muchos casos produjeron desvíos en los tiempos del proyecto.

Figura 6-8: evolución de los riesgos a lo largo del tiempo

En esta grafica podemos visualizar la desaparición de un riesgo en etapas tempranas al proyecto, este correspondía al trabajo a la distancia que se presentaba por uno de los integrantes que no se encontraba viviendo en el mismo lugar que el resto. Cuando esto se terminó, el riesgo dejo de estar presente en el proyecto.

Luego el primer riesgo que presentó un aumento radical que significo la atención del equipo para su mitigación fue la problemática de la conversión de la base de datos del cliente, la misma se terminó resolviendo implementando una interfaz en donde lo que se buscaba era establecer un protocolo de comunicación donde ambas partes entendieran el modelo de datos.

Otro riesgo que se presentó y significo una re planificación de ciertas tareas, fue el uso de base de datos en columnas, el que presentó un contratiempo para el equipo teniendo que cambiar de tecnología adoptada.

De los riesgos críticos el que se tardó más en mitigar fue el de poder lograr una interfaz intuitiva. Llegando al final del proyecto habiendo disminuido el riesgo, pero no pudiendo mitigarlo del todo. El mismo fue tomando importancia con el paso del tiempo, aumento también la probabilidad de ocurrencia a medida que pasaba el tiempo y se recibían evaluaciones por parte de usuarios.

7 Construcción del *software*

El presente capítulo describe cómo se llevó adelante la construcción del producto y algunas decisiones que se tomaron a lo largo de ésta etapa y que fueron clave para la obtención del producto final.

7.1 Principales decisiones tomadas

Dados los requerimientos cambiantes del *software* y grandes desafios tecnológicos, se debieron utilizar mecanismos de reducción de complejidad.

7.1.1 Recepción de datos desde PSIG

Uno de los componentes más importantes y de mayor incertidumbre iniciales fue el que recibiría los datos de PSIG para realizar reportes. Porque no se contaba con una estructura definida para los datos que los usuarios luego consultarían y PSIG no contaba con una forma sencilla de reconstruir una transacción desde su base de datos.

La realidad de PSIG es que cuenta con una cartera de clientes muy amplia y variada, esto impactaba en que se debieron contemplar múltiples rubros e innumerables particularidades de cada cliente de PSIG.

También al PSIG brindar soluciones a medida customizadas para sus clientes agregaba aún más complejidad a la hora de definir una transacción estándar. Por ejemplo en diversos clientes del mismo rubro los datos que reflejaban un aspecto de la realidad diferían entre cliente y cliente.

Pero se contaba con una muy buena disposición de su parte para abordar una solución que solucionase este problema complejo.

Por lo tanto para recibir los datos de PSIG se detectaron tres opciones posibles:

1. Consumir directamente desde la base de datos de PSIG los datos
2. Que PSIG expusiera una interfaz que provee los datos
3. Que el equipo expusiera una interfaz para que PSIG nos enviará los datos

La primera opción se vio descartada después de contar con una instancia de la base de datos y ver que era sumamente costoso y conllevaría un esfuerzo en horas muy superior al esperado y

capaz de realizar durante el proyecto para esta tarea. Por ejemplo, en cada instalación de PSIG los mismos datos de la realidad eran guardados y/o estructurados de forma distinta, no se contaba con una lógica clara para poder re estructurar una transacción y a nivel de código se contaba con un *software* legado (de hasta 40 años en algunos casos) el cual contenía la lógica para poder re construir una transacción. A partir de estos últimos ejemplos quedó claro que tratar de re construir una transacción a partir de la base de datos de PSIG no era viable por más que fuese la mejor opción a nivel de performance para sincronizar datos entre sistemas.

Luego de una búsqueda bibliográfica, consultas a expertos y conocimiento de las particularidades de PSIG en cada instalación, se decidió que la mejor opción era desarrollar el equipo una interfaz para que PSIG enviara los datos de sus transacciones.

Recordando que según lo acordado para el alcance del proyecto solamente se trabajaría con datos de compras y ventas, nos basamos en el trabajo de [19] para justificar que no solo tiene sentido que el equipo haya implementado la interfaz para consumir datos de PSIG, sino que también de esta forma el equipo tiene mayor control sobre los datos a trabajar y disminuye el impacto de cambio, siendo el propietario de un contrato teóricamente estable.

Otro factor que influyó en que el equipo implementara la interfaz fue que el cliente no quería tener la responsabilidad de implementar esta, ni de tener que definirla. Una vez definida la interfaz se validó con el cliente y se ajustaron algunos factores que solicito para que el envío de información fuese más sencillo para ellos.

En resumen se hizo uso del principio de inversión de dependencias para disminuir el impacto de cambio de la lógica de negocio de PSIG en los datos con los que el equipo trabajó para realizar los reportes. Y se tomó una decisión que dadas las características del cliente y la gran variedad de rubros con los que trabaja, resuelve un problema complejo de una forma completa y efectiva.

7.1.2 Traducción de consultas

En general los sistemas de generación de consultas suelen ser abordados como un problema conocido como "*query by example*". Este mismo implica que se debe dar a conocer al usuario la completitud del dominio de los datos; y para favorecer la usabilidad del mismo es

necesario proveer al usuario con el dominio completo del sistema de una forma amigable e intuitiva de ser posible esta última. Esto es considerado un problema complejo.

Para cumplir con este requisito se diseñó una interfaz de usuario que permite trabajar con todos los datos del dominio, de una forma amigable e intuitiva para el usuario. Pero el usuario no debe conocer la estructura interna de los datos para realizar consultas.

En concreto se utilizó una base de datos entre la base de datos en columnas y la interfaz de usuario para poder mapear los datos del dominio a datos que el usuario captara como naturales para él. Esto sería comparable con un cubo en las soluciones actuales planteadas en tecnología OLAP.

Estas características son tomadas de la BD en columnas implementada con Infobright (ver sección arquitectura), que como ya se explicó en el subtítulo “Recepción de datos desde PSIG” cuenta con los datos necesarios para que el usuario pueda realizar todas las consultas. Para más detalles de cómo se realizó el mapeo de los datos consultar el anexo de modelos conceptuales de compras y ventas.

7.1.3 Base de datos en columnas a utilizar

El equipo debió enfrentarse a un problema complejo debido a la decisión de utilizar MonetDB como base de datos en columnas. Este fue la integración con Groovy on Grails.

Como fue mencionado en el capítulo de gestión del proyecto, se decidió utilizar MonetDB como base de datos en columnas ya que presentó los mejores tiempos de lectura y disponía de un driver desarrollado en Java, por lo que no pareció representar problemas al integrar con Grails. Pero finalmente el driver en Java no se encontraba en óptimas condiciones ni era utilizado por nadie en la comunidad de Grails, esto ocasionó grandes demoras en la integración de MonetDB con Grails. Para poder hacerlo, el equipo debió recurrir a diversos foros, comunidad de MonetDB, comunidad de Grails y consultar a expertos en ambas tecnologías.

Luego de lograr conectividad entre ambas tecnologías y evaluar los resultados de la información recopilada, el equipo llegó a la conclusión que mantener esa conexión y poder implementar todas las funcionalidades necesarias sería excesivamente trabajoso. Por lo tanto

el equipo retomó la segunda opción de base de datos en columnas (Infobright), la cual fue sumamente fácil de integrar y utilizar.

Las razones de por qué sería sumamente trabajoso y riesgoso mantener la conexión con MonetDB son:

- En la comunidad de MonetDB se encontraron muchos bugs asociados a performance.
- En la comunidad de Grails no se contaba con un *plugin* para incorporar MonetDB como *data source* y por lo tanto se debía implementar toda la lógica necesaria para utilizar el driver de Java. También el equipo intentó desarrollar un *plugin* propio y subirlo al repositorio de *plugins* de Grails, pero debido a que los *plugins* deben pasar una etapa de aceptación de tiempo indefinido, se abandonó esa alternativa.
- Los expertos no recomendaron seguir dependiendo de una tecnología con tantos riesgos, en un sistema donde el correcto funcionamiento de dicha tecnología es crucial.

7.1.4 Servidor a utilizar

Amazon Elastic Compute Cloud (Amazon EC2) es uno de los productos ofrecidos por la plataforma de cómputo en la nube de Amazon llamada Amazon Web Services (AWS). EC2 ofrece y permite rentar máquinas virtuales, en donde se puede instalar el sistema operativo que se desea. Se visualizó en este producto grandes ventajas, entre ellas:

- Según datos de recolectados por el sitio CloudHarmony [20], ofrece disponibilidad teniendo un *downtime* de 29,2 segundos en 30 días, dando una disponibilidad del 99.9989%.
- Fácil escalabilidad de sus productos y se paga solo por la capacidad utilizada.
- Fácil y buena interacción entre distintos productos utilizados de la plataforma de AWS.
- Máquina virtual con Windows Server 2008 gratuita por un año.

Para este caso se rentó una instancia Windows Server 2008 ya que el mismo sistema operativo es el utilizado por el cliente en sus servidores.

7.1.5 Anticipación al cambio

Construir *software* que se anticipe al cambio es una tendencia muy usual hoy en día, pero en concreto para este proyecto resultaba necesario anticiparnos a posibles cambios la lógica del negocio de nuestros clientes (Ej: empresas de pequeño o gran porte que comienzan a generar nueva información dado su crecimiento).

Si bien se acordó que solo se trabajaría con información de compras y ventas, la interfaz que recibe los datos de transacciones es capaz de recibir datos asociados a otros tipos de transacciones con un mínimo esfuerzo y desarrollo. Se debe enviar al comienzo del JSON el nombre del concepto asociado a la transacción y acordar previamente entre las partes un formato estándar para este concepto (ej: información contable). Solo con esto el equipo podrá analizar y definir cuál es la manera más conveniente de almacenar la información de este nuevo concepto y con un par de líneas de código agregar este tipo de transacciones a las ya existentes. Luego para permitirle minar información al usuario sobre estos nuevos datos no resta más que definir reglas de mapeo entre los datos guardados y los datos que se le mostrarán al usuario (Ver anexo Mapeo de datos del dominio y datos para el usuario, sub título “Futuros Mapeos”).

7.1.6 Usabilidad

Como ya se menciona en el capítulo Arquitectura de software, el diseño y gestión de la usabilidad de UI fue un factor crítico en el sistema. Se utilizaron mecanismos para plantear una UI que satisficiese los altos estándares esperados por parte del cliente y usuarios finales. Pero también fue necesario gestionar la evolución y tasa de aceptación de la usabilidad a lo largo del proyecto.

A mitad del ciclo de desarrollo gracias a pruebas con usuarios finales se pudo detectar la UI no estaba cumpliendo con los estándares definidos, y que esta principalmente no era intuitiva para el usuario. En concreto se estaba sobrecargando demasiado con información y la principal pregunta que se planteó fue: ¿Cómo es posible mejorar la experiencia del usuario con la UI sin quitarle funcionalidades al mismo?

Para poder generar una UI más sencilla e intuitiva se debió volver a la definición requerimientos, el equipo con ayuda del cliente logró definir cuáles eran los aspectos claves en los que debía enfocarse la UI.

Estos eran:

1) En su gran mayoría los reportes generados sumarán cantidades o montos, la UI debería contemplar esta premisa.
2) Los reportes generados no deben mostrar en primera instancia todos los datos existentes, esto genera una sobrecarga de datos y evita que el usuario se pueda enfocar en la información que quiso consultar.
3) No hacer foco en los filtros, estos pueden ser vistos como cortes horizontales en un Excel según el cliente.

En base a estas tres conclusiones y nuevos modelos conceptuales de UI se desarrolló una nueva UI que satisficiese tanto los requerimientos iniciales como los emergidos.

La nueva interfaz se presenta a continuación:

Figura 7-1: interfaz de usuario actual para crear el reporte.

7.2 Construcción y verificación

Al final de cada iteración se buscó tener un entregable de valor para el cliente, como así define FDD. Y por lo tanto se fue a validar con el cliente.

7.2.1 Estándares y construcción

Se siguieron los estándares de codificación en cada lenguaje utilizado (Grails y PHP). [21] [22]

7.3 Gestión de la construcción

7.3.1 Planificación de la construcción

El equipo se reunía al final de cada iteración y luego de una reunión con el tutor, se re definía el alcance de la próxima iteración en cuanto a *features* a desarrollar. Esto principalmente era necesario disminuir la cantidad de puntos a realizar.

7.3.2 Tecnologías de construcción

Los lenguajes utilizados para la construcción del *software* fueron PHP, Groovy on Grails, JavaScript, jQuery, CSS3, HTML5, MySQL. A continuación un mapeo de los artefactos de la arquitectura con las tecnologías.

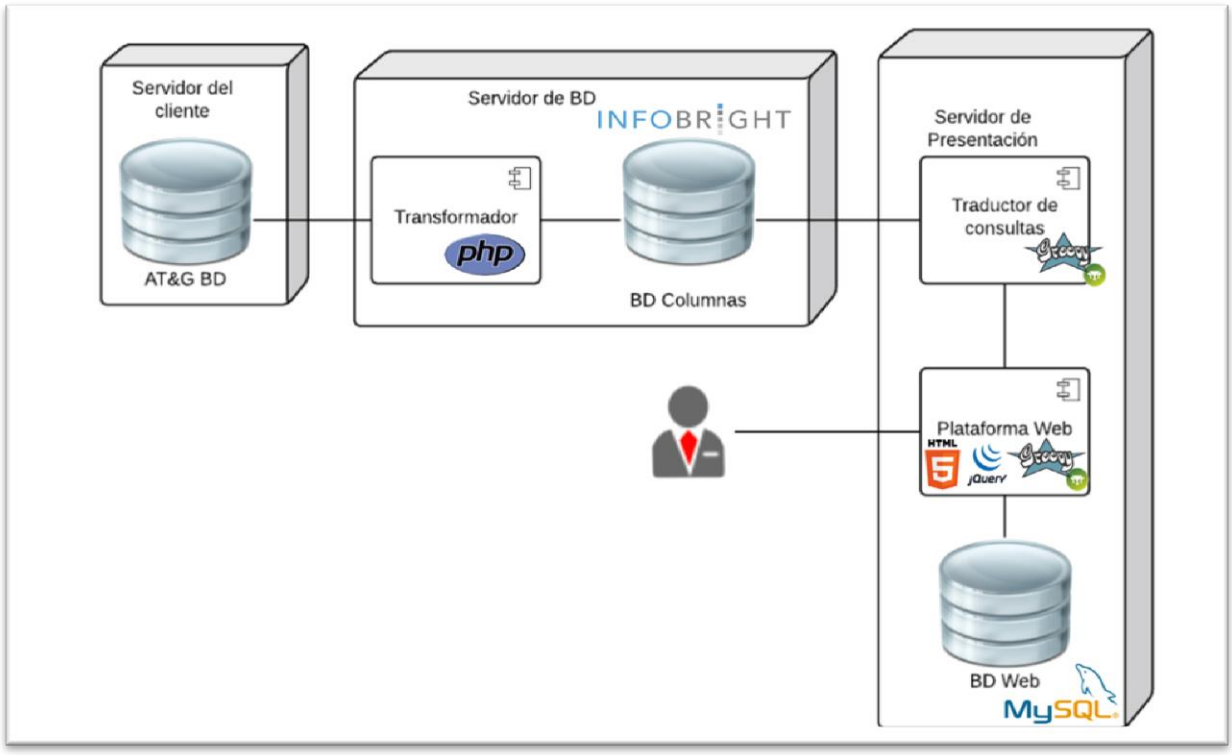

Figura 7-2: mapeo de las tecnologías en la arquitectura

Para la base de datos en columnas se utilizó Infobright por lo mencionado en la sección de que base de datos utilizar. Para el módulo que recibe información desde PSIG y luego la inserta en la base de datos, se usó PHP.

Para la lógica de tanto el traductor de consultas como la plataforma web se utilizó Groovy on Grails. Para la parte web se utilizó HTML5 para la estructura, CSS3 para la apariencia y JavaScript y jQuery para el comportamiento. La base de datos que guarda la información de los usuarios se hizo en MySQL.

7.3.3 Diseño y Uso de APIs

Dentro del ecosistema del *software* se utilizaron RESTful APIs para comunicar componentes. Se utilizó este tipo de APIs y no SOAP ya que son más flexibles, dejan mejor expresado que recurso se está exponiendo y consumiendo si se utilizan correctamente los estándares.

7.3.4 Entornos de Desarrollo

Se utilizaron distintos entornos de desarrollo dentro del equipo: Sublime Text 2 y el IntellIj IDEA. La decisión de cual usar quedó en manos de cada miembro del equipo.

Como repositorio de código se utilizó BitBucket y se gestionaba el código utilizando GIT a través del programa SourceTree.

7.3.5 Herramientas de Testing Unitario

Se utilizó JUnit que ya está incorporado en el *framework* de desarrollo Groovy on Grails.

7.3.6 Perfiles, Análisis de Performance

Se utilizaron los respectivos clientes de cada proveedor de bases de datos orientadas a columnas para medir la performance de las consultas en etapas iniciales del proyecto. Y finalmente mediante *logs* se midió la performance de las mismas a medida que avanzaba el proyecto.

7.4 Cambio de alcance

En las etapas finales del proyecto, el cliente se mostró disconforme con la calidad de la interfaz de usuario. El resultado fue un aumento en el alcance del producto ya que atender los cambios que implicaba la disconformidad constaba en básicamente rehacer la interfaz de usuario y como se maneja la información que ésta genera. También se plantearon nuevos requerimientos, los cuales no fueron incluidos debido a que implicaba más tiempo del que el equipo disponía para el proyecto.

Se acordó con el cliente la mejora de la UI y de cómo se presenta la información a los usuarios finales para cumplir con la calidad esperada, dejando los nuevos requerimientos que surgieron para un post proyecto.

El equipo debió volver a las bases de un reporte y en conjunto con el cliente y reuniones con usuarios finales se ideó una nueva UI más enfocada, sencilla y entendible.

Si bien el equipo tuvo la posibilidad de buscar nuevos clientes y evitar este cambio de alcance, debido al buen trato y continua disposición durante el proyecto por parte del cliente actual, se decidió continuar de todas formas. También vale destacar que dada la evolución del producto gracias a este cambio de alcance, el equipo tiene aún más confianza y puede afirmar que se logró un producto fácil de usar, intuitivo y capaz de generar información útil para los usuarios.

8 Gestión de la calidad del *software*

Éste capítulo profundiza en cómo el equipo definió, planificó y controló la calidad a lo largo del proyecto. Comienza describiendo objetivos y actividades, muestra resultados de las evaluaciones y por último las conclusiones alcanzadas.

8.1 Objetivos de calidad

La calidad del producto es uno de los aspectos más importantes del mismo. Un producto de baja calidad es un producto que probablemente no se use, no sirva para lo que fue pensado, o no satisfaga al cliente.

Es por esto que al comienzo del proyecto el equipo definió un conjunto objetivos de calidad con los que el producto debe cumplir. Para que el producto alcance dichos objetivos, es necesario un proceso de buena calidad, por lo tanto el equipo definió también objetivos de calidad para el proceso.

A continuación se detallan los objetivos de calidad tanto para el producto como para el proceso.

8.1.1 Producto

Ausencia de defectos

Se busca alcanzar un producto sin defectos cuando llegue a manos del cliente. Lo mismo implica una gran inversión en pruebas y calidad en el desarrollo.

Interfaz gráfica intuitiva

Como objetivo de la interfaz gráfica se plantea lograr que los usuarios puedan utilizar el 80% de las funcionalidades del producto sin necesidad de una capacitación del mismo. Se decidió este porcentaje siguiendo el principio de Pareto. [23] Apuntando a la mejora continua en este aspecto.

8.1.2 Proceso

Mejora continua

La calidad del producto está fuertemente ligada a la calidad de un proceso, por eso es importante que dicho proceso sea de calidad. Es por esto que la mejora del proceso es clave en la obtención de un producto que satisfaga al cliente. La obtención de métricas luego de cada iteración es imprescindible para poder encontrar los aspectos a mejorar del proceso y así obtener un mejor producto al final del proyecto.

Proceso adecuado

Definir un proceso que sea adecuado a las características del proyecto y del equipo. Buscando favorecer y ayudar al equipo a realizar el mismo.

8.2 Aseguramiento de la calidad

Según Ian Sommerville, "el aseguramiento de la calidad es el proceso de definir como la calidad del *software* puede ser alcanzada y como los encargados del desarrollo saben que el *software* tiene el nivel requerido de calidad" [24]. Para asegurar la calidad, el equipo definió al comienzo del proyecto, un plan de calidad que contempla cada etapa del mismo, y un conjunto de estándares que hicieron más fácil el alcanzar los objetivos de calidad.

Estándares

También Sommerville define que existen dos tipos de estándares, los aplicables al producto y los aplicables al proceso. [24]

Los aplicables al producto refieren a estándares de documentación, documentación de requerimientos por ejemplo, estándares de codificación y estándares de cómo un lenguaje de programación debe ser usado. Los estándares de proceso refieren a la definición de proceso de especificación, de diseño y de validación, y la descripción de los documentos que deben ser escritos a lo largo de dichos procesos.

Estándares de producto

- Diseño web adaptable a todo tipo de dispositivo.

Como estándares de codificación, el equipo definió:

- Código en inglés
- Nombres de las variables y métodos deben ser nemotécnicos

- Realizar comentarios descriptivos en el código
- Mantener espacios entre métodos de forma de mejorar la legibilidad

Estándares del proceso

El equipo consideró la comunicación con el cliente una parte clave para dar con el producto correcto, por lo que definió estándares de cómo documentar dicha comunicación. Para consultas puntuales, el equipo determinó que el *Domain Manager* sería quien las llevaría a cabo, vía mail.

Para las reuniones con el cliente, se creó un molde con la finalidad de documentar correctamente cuál es el objetivo de la reunión, quienes son los participantes, que se discutió y a que conclusiones se llegaron.

Plan de calidad

El equipo creó al comienzo del proyecto un plan de calidad, con el fin de definir las fases y las actividades de calidad a realizar durante el proyecto. También define los estándares anteriormente mencionados. El plan de calidad se puede ver en el anexo Plan de calidad.

8.3 Prueba de *Software*

Las pruebas del *software* se llevaron a cabo durante toda la etapa de construcción. El proceso de las mismas, se detalla a continuación:

Pruebas del desarrollador

Las mismas son realizadas por cada desarrollador en el transcurso de cada iteración. Las mismas deben abarcar casos válidos, inválidos y casos borde.

Pruebas de validación

Estas son realizadas una vez finalizada cada iteración y lo que se prueba es el producto resultado de la misma. Se busca determinar que el software entregado este funcionando de la forma esperada.

Pruebas de integración

Las mismas se realizan cada vez que se integran desarrollos hechos por cada integrante del equipo. En las mismas se controla se mantenga cada prueba realizada por cada desarrollador, de forma de determinar que la integración no afecto el funcionamiento individual de cada desarrollo.

Pruebas de regresión

Las mismas deben realizarse en la fase de cierre del proyecto, y se ocupan de verificar que no se haya introducido algún defecto. Durante las mismas también se verifica el correcto funcionamiento de la totalidad de las funcionalidades del producto.

Pruebas de usuario

Estas pruebas son realizadas en conjunto con usuarios, los cuales prueban el sistema devolviendo opiniones y *feedback* tanto directo como indirecto, al decir indirecto se refiere a las interpretaciones que toma el equipo sobre acciones, actitudes y gestos que produjo el usuario durante la prueba.

En las mismas se observan distintos aspectos del producto: tiempo que insume realizar ciertas tareas específicas, tiempo de aprendizaje del producto, cantidad de ayuda que se le debe brindar al usuario, facilidad de uso, esfuerzo del usuario para el uso del producto y primer impacto sobre la interfaz gráfica.

Estas pruebas intentan realizarse tanto en futuros usuarios (usuarios de empresas que utilizaran el sistema), potenciales usuarios (usuarios con posiciones similares en empresas que de momento no mostraron interés en el producto). Como potenciales usuarios el equipo definió contactar a todo usuario que pertenezca al área de ventas de cualquier empresa.

8.4 Evaluación

Para la evaluación del sistema, se definieron e utilizaron muchas medidas, las cuales serán descriptas en esta sección.

Puntos

Cada *feature set* tiene un tamaño en puntos, el cual lo determinó el equipo en una primera instancia. Vale destacar que un punto es una unidad de medida que fue adoptada por el

equipo y fue adquiriendo un valor más exacto con el transcurso del proyecto, ese valor no está relacionado a ninguna otra medida conocida.

Por lo que, cuando se habla de los puntos realizados, se corresponde a la cantidad de esos puntos que fueron completados, ya sea en una iteración, una semana, día, etc.

Defecto

A la hora de definir un defecto, el equipo decide adoptar la definición proporcionada por la IEEE en donde define a un defecto como "una imperfección o deficiencia en un producto donde ese producto no alcanza los requerimientos o especificaciones y necesita ser reparado o cambiado". [25]

Líneas de código

Es una medida que indica la extensión del código fuente del sistema en cuestión. La misma se halla con la herramienta Metrics Reloaded.

Esfuerzo

Esfuerzo para este equipo, se corresponde a la cantidad de horas invertidas en una tarea.

Capacidad

Es una medida del equipo que proporciona la cantidad de horas disponibles para invertir en una tarea.

8.5 Métricas

El principal objetivo de la recolección de métricas es el medir, controlar y mejorar el proceso. Cada métrica permite analizar eficiencia y efectividad del equipo durante el proyecto, haciendo uso de los indicadores que poseen.

Durante el proyecto, fueron utilizadas para la mejora de procesos, ayudando también a la toma de decisiones del equipo ofreciéndole a estos medidas cuantitativas sobre las cuales basar sus convicciones.

Dentro del proyecto se detectaron dos tipos de métricas:

- Métricas de producto
- Métricas de proceso

8.5.1 Métricas

La primera métrica a analizar es el avance del alcance por iteración. En esta métrica se presenta el avance del proyecto luego de cada iteración en cuanto a lo realizado y lo que resta hacer. En la figura 8-1 se puede visualizar un avance lineal con respecto a lo realizado en cada iteración, pudiéndose ver que la magnitud de los puntos que el equipo se compromete en cada iteración no varía significativamente. Mientras que se puede observar que el crecimiento de los puntos a realizar en cada iteración no sigue la misma lógica ya que los mismos se agregaban en ciertas instancias específicas, incluyéndose estos nuevos puntos en la iteración próxima a comenzar.

En la iteración número 9 se puede visualizar un gran incremento en los puntos totales del proyecto, esto se debió que en la misma se incluyeron nuevos *features* que involucraban cambios tanto en la interfaz gráfica exigida por el cliente, como en la tecnología utilizada para la base de datos en columnas, entre otras cosas.

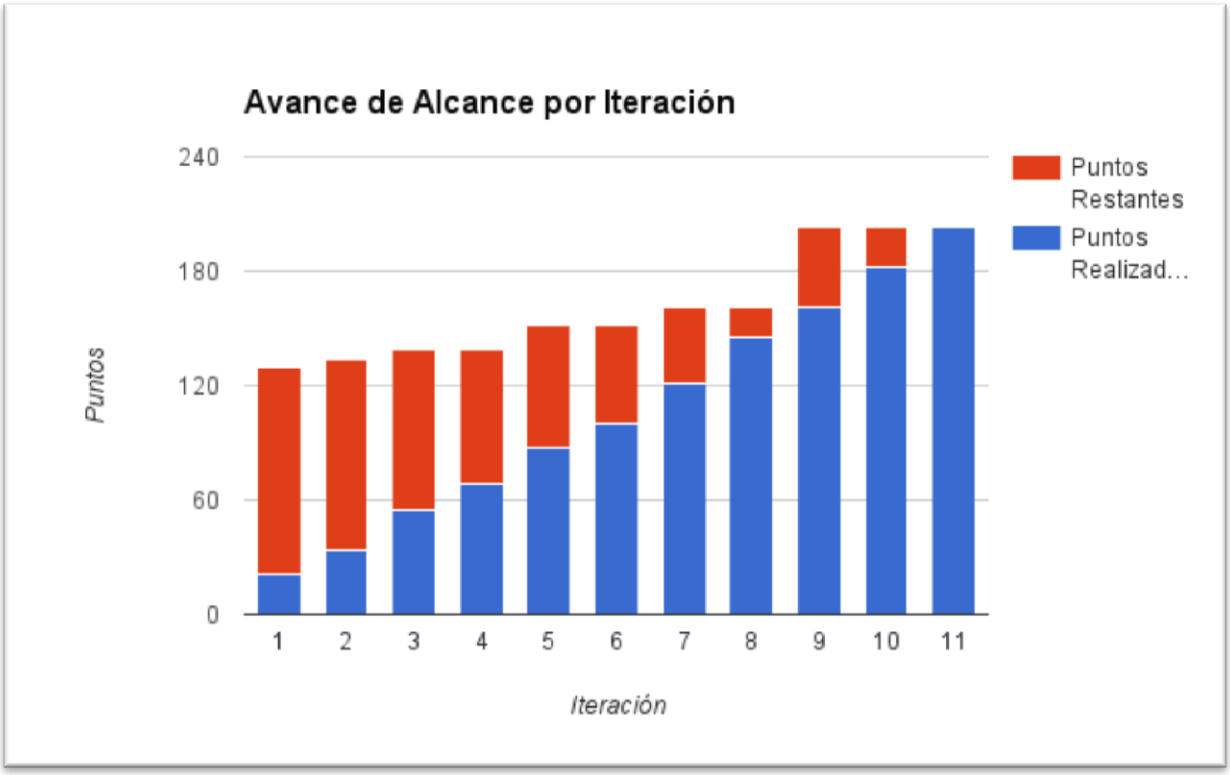

Figura 8-1: avance del alcance por iteración

La siguiente métrica corresponde a la velocidad del equipo. La misma se definió como puntos realizados en cada iteración por esfuerzo (horas) que involucro la realización de los mismos. Como se observa en la figura 8-2, hay un leve aumento de velocidad en la tercer iteración, que se interpreta como un mejor manejo de las tecnologías por parte del equipo, sumado a que el *feature* implicado en dicha iteración tuvo un desarrollo más rápido de lo pensado.

En la cuarta iteración se produjo un declive en la velocidad producto del problema surgido con tecnología adoptada para implementar la base de datos en columnas. El mismo produjo que se consuman horas en ese *feature* no pudiendo terminarlo y teniendo que pasarlo a la siguiente iteración, no teniendo una buena productividad en dicha iteración, esto también se puede visualizar en la anterior grafica en donde el avance realizado en la cuarta iteración fue menor al resto. Luego lo mismo genero ventajas en la siguiente iteración, teniendo algunos avances frente al *feature* que involucraba a la base de datos en columnas, esto hizo aumentar levemente la productividad del equipo.

En la sexta iteración se puede observar una leve disminución en la velocidad producto de un defecto en la interfaz donde el cliente se comunica a nosotros y nos envía sus transacciones, esto nos generó demoras en la realización de los *features*.

Por último la última disminución de velocidad se produjo por el cambio de alcance que se realizó en dicha iteración.

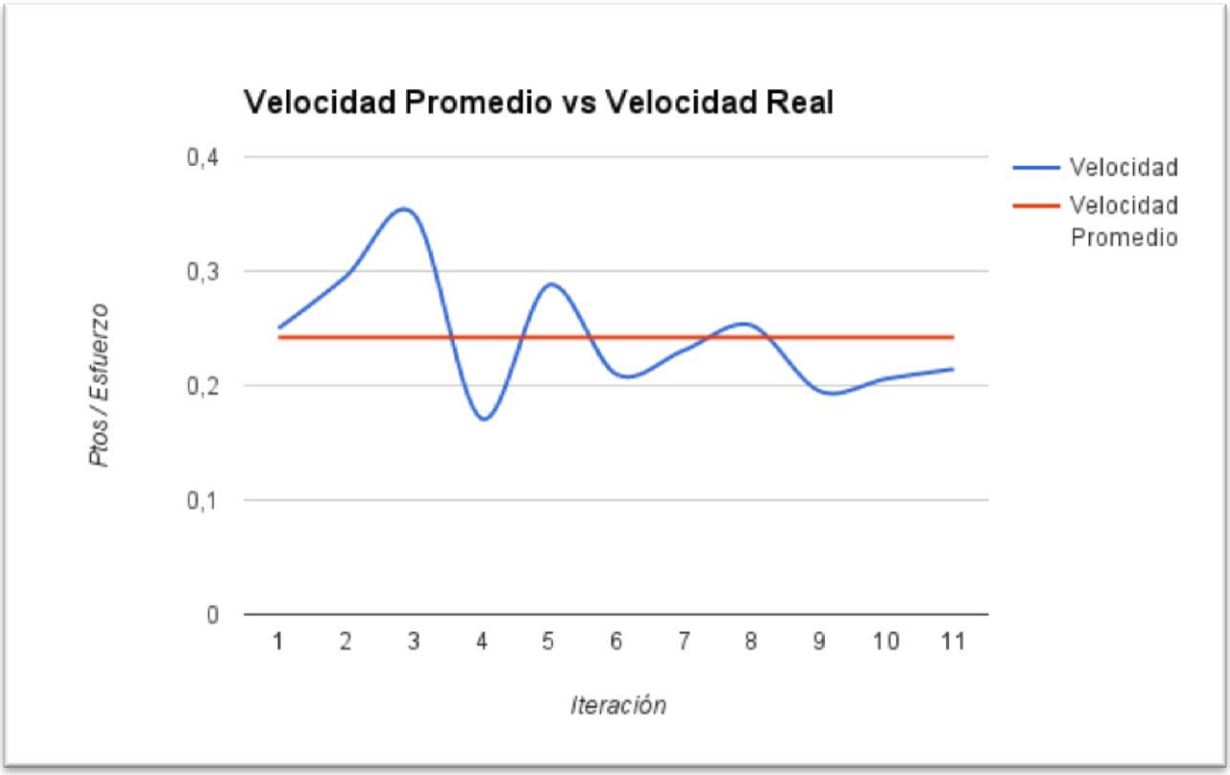

Figura 8-2: gráfica de velocidad promedio contra velocidad real

Por último, la métrica de capacidad y esfuerzo por cada iteración. Como se ve en la figura 8-3, existe un aumento del esfuerzo respecto a la capacidad en las últimas iteraciones que fue cuando el equipo invirtió más tiempo de desarrollo para terminar el producto en tiempo y forma. Los aumentos o decrementos del esfuerzo del equipo a lo largo del proyecto responden a las otras actividades académicas o laborales en la que los miembros del equipo estaban comprometidos.

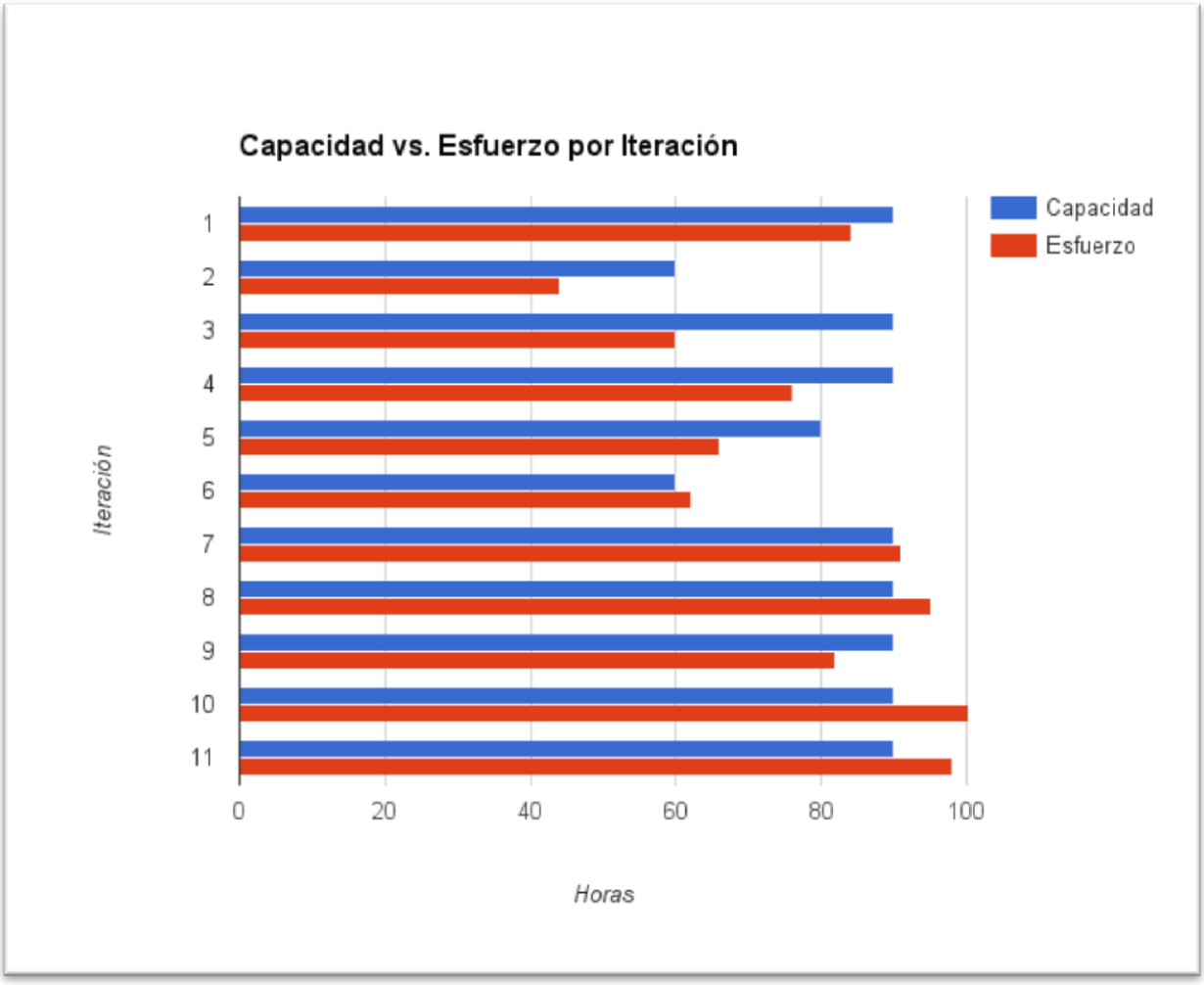

Figura 8-3: grafica de capacidad y esfuerzo

8.5.2 Usabilidad

En cuanto a usabilidad fue un atributo de calidad al que se le fue dando mayor importancia a medida que el proyecto avanzaba. El equipo observo que el mismo no era tan fácil como pensaba, y el control de la calidad sobre el mismo comenzó en etapas más tardías del proyecto, después de recibir *feedbacks* negativos en las primeras pruebas de usuario, no habiéndolo considera al mismo como importante dentro de la gestión de la calidad previamente.

Luego de sucedido esto se realizaron reuniones con el cliente y potenciales usuarios, para definir cambios en la interfaz, que significaron una reestructuración en la misma. Para la coordinación de las mismas se utilizó la herramienta de Google Calendar en donde se buscó definir los aspectos a tratar en dicha reunión.

Dentro de las pruebas de usuario, que están incluidas en más detalle dentro de las minutas del proyecto que se encuentran en otro directorio de la entrega, se encuentran estas dos tablas resultantes de dos pruebas que se obtuvieron de usuarios que probaron la primer interfaz gráfica del sistema.

Estudio cuantitativo del usuario Johnny Walker sobre la primera interfaz gráfica del sistema:

	Generar reporte	**Ver reporte**
Amigable (1-5)	2	3
Intuitivo (1-5)	1	4
Lindo (1-5)	3	4
Ordenado (1-5)	3	4
Fácil de usar (1-5)	1	3
Claridad (1-5)	1	3
Útil (1-5)	4	4

Tabla 8-1: estudio cuantitativo del usuario Johnny Walker sobre la primera interfaz gráfica del sistema

Estudio cuantitativo del usuario Rodrigo Risso sobre la primera interfaz gráfica del sistema:

	Generar reporte	**Ver reporte**
Amigable (1-5)	1	3
Intuitivo (1-5)	1	4
Lindo (1-5)	2	3
Ordenado (1-5)	2	4
Fácil de usar (1-5)	1	4
Claridad (1-5)	2	4
Útil (1-5)	4	4

Tabla 8-2: estudio cuantitativo del usuario Rodrigo Risso sobre la primera interfaz gráfica del sistema

Estudio cuantitativo del usuario Johnny Walker sobre la interfaz gráfica final del sistema:

	Generar reporte	Ver reporte
Amigable (1-5)	4	4
Intuitivo (1-5)	3	5
Lindo (1-5)	4	4
Ordenado (1-5)	4	4
Fácil de usar (1-5)	4	5
Claridad (1-5)	4	5
Útil (1-5)	4	4

Tabla 8-3: estudio cuantitativo del usuario Johnny Walker sobre la interfaz gráfica final

Estudio cuantitativo del usuario Rodrigo Risso sobre la interfaz gráfica final del sistema:

	Generar reporte	Ver reporte
Amigable (1-5)	3	4
Intuitivo (1-5)	3	4
Lindo (1-5)	4	4
Ordenado (1-5)	4	5
Fácil de usar (1-5)	3	5
Claridad (1-5)	3	4
Útil (1-5)	5	5

Tabla 8-4: estudio cuantitativo del usuario Rodrigo Risso sobre la interfaz gráfica final

Referencia de las tablas: 1 es malo y 5 excelente.

Se puede observar un avance entre los comienzos de la interfaz gráfica y lo que se logró al final del proyecto.

8.6 Conclusiones

La gestión de la calidad fue un aspecto el cual se mejoró mucho si se compara el principio del proyecto con el final, en el comienzo no se corroboraba correctamente la calidad de lo que se producía, aprendiendo luego de contratiempos y re trabajo producto de la mala gestión de la misma, lo que derivó en dar mayor foco a dicha gestión.

Las mejoras se produjeron en principio al comenzar a cumplir lo comprometido en cantidad de horas invertidas en alcance, cumpliendo el principio de FDD que dice que el alcance dentro de una iteración no debe variar, pudiendo variar si la cantidad de horas, o esfuerzo, que se dedicó a la misma. Esto mejoro la calidad de los productos resultantes de cada iteración, debido a que previamente la disminución de horas de alcance por iteración impactaba directamente en la calidad, por disminución de tiempo de prueba entre otras cosas.

Otros aspectos en los que se pudo ver fue en los resultados de las pruebas finales de usuarios, en donde se visualiza una mayor conformidad de los usuarios frente a la interfaz gráfica.

8.7 Revisión de objetivos

Ausencia de defectos

En cuanto a este objetivo, se considera que no es suficiente la cantidad de pruebas que se ha hecho sobre el producto para considerar que el mismo tiene un funcionamiento confiable, libre de defectos que afecten al flujo básico de la aplicación. Por lo tanto el equipo debería continuar desarrollando pruebas para lograr el objetivo.

Interfaz gráfica intuitiva

Este objetivo está cumplido, habiendo una mejora final de la interfaz gráfica que permitió observar, en las pruebas a usuarios, que la cantidad de funcionalidades que el usuario podía realizar sin capacitación llegaba al 80%.

Mejora continua

El equipo considera que este objetivo se cumplió, el proceso sufrió mejoras a lo largo del proyecto que lograron que el mismo se adecue mejor al equipo, así como también se destaca

que el equipo fue aprendiendo del proceso pudiendo ejecutarlo de mejor manera. Las métricas fueron importantes para que el equipo aplicara mejoras al proceso.

Proceso adecuado

Por último, este objetivo se considera cumplido, pudiendo observar que el proceso definido ayudó al equipo en la realización del proyecto y se adecuo a las características del mismo. Esto no quita que el equipo sufrió un proceso de adaptación de dicho proceso.

9 Gestión de configuración del *software*

La gestión de configuración del *software* (SCM) controla la evolución e integridad de un producto mediante la identificación de sus elementos; la gestión y el control de cambios; verificación, registro y generación de reportes sobre la información de configuración. Esto a su vez, requiere una comprensión del contexto de la organización, y las restricciones impuestas en el diseño e implementación del proceso de SCM.

La gestión del proceso de SCM fue liderada por Joaquín Colella, siendo de igual manera todos partícipes de las actividades que la misma demandaba. Estas fueron establecidas y comenzaron a realizarse a comienzos del mes de Abril del 2015.

Es importante destacar, que la SCM es una actividad de protección, la cual se aplica durante todo el ciclo de vida, donde se identifica, controla audita e informa las modificaciones de los elementos del *software*. Uno de los principales problemas detectados en el análisis es la falta de una política de control de cambios, y sobre todas las cosas controlar las versiones de desarrollo de *software*.

9.1 Identificación de la configuración de *software*

La identificación de *software* identifica los elementos que deben ser controlados, establece un esquema de los elementos y sus versiones, y establece las herramientas y técnicas que se utilizan en la gestión de los productos controlados. Estas actividades proporcionan la base para el resto de responsabilidades de SCM.

En este proyecto se encuentran tres tipos de ítems de configuración de *software*:

- Documentos: los cuales tienen tanto relación directa con el desarrollo, como es el caso de los *feature sets*, así como indirecta como es el caso del contrato firmado con el cliente.
- *Software*: código fuente de los módulos que conforman a la plataforma desarrollada.
- Datos: datos de prueba, del cliente y esquemas de las bases de datos.

En el anexo Elementos de configuración de *software* se presenta los elementos que deben ser controlados.

9.2 Control de la configuración de *software*

El control de la configuración del *software* refiere a la gestión de cambios en el *software* durante el ciclo del mismo. Cubre el proceso para determinar qué cambios hacer, que autoridades deben aprobar dichos cambios, el soporte a la implementación de los cambios, y el concepto de desviaciones formales de los requerimientos del proyecto, así como exenciones de ellos. La información derivada de estas actividades es útil para medir el tráfico de los cambios así como aspectos del re trabajo.

En el caso de nuestro proyecto se utilizó en todo momento un sistema colaborativo, el cual no bloqueaba un elemento frente a la posible modificación del mismo por parte de otro. Este esquema se adoptó tanto en los repositorios como en la documentación, estando la misma disponible en todo momento a todos.

Cada módulo cuenta con un responsable el cual se encarga de autorizar un cambio, lo que significa que esa misma persona debe estar al tanto de todo desarrollo que se está produciendo en ese módulo del cual es responsable. Una vez que se está autorizado se sube el cambio.

Se aplicaron desviaciones formales en algunos repositorios, en casos donde el cambio era tan grande que ciertas cosas ya realizadas corrían riesgo de tener problemas frente al nuevo cambio.

Se estableció en un principio ofrecer una sola versión generalizada para todos los clientes, no debiendo tener que llevar control de estas posibles diferencias.

9.3 Revisión de la configuración de *software*

Se pactó internamente en el equipo en realizar ciertos controles internos donde un integrante del mismo controla el trabajo realizado por otro y si el mismo cumple con lo previsto, a pesar de no contar como auditoría, ayudó a detectar problemas en etapas tempranas.

9.4 Gestión de *software* de entregas y liberaciones

En este contexto, la liberación se refiere a la distribución de un elemento de la configuración de *software* fuera de la actividad de desarrollo; esto incluye lanzamientos internos, así como la distribución a los clientes. Cuando diferentes versiones de un elemento de *software* están

disponibles para la entrega (como versiones para diferentes plataformas o versiones con diferentes capacidades), es con frecuencia necesario volver a crear versiones específicas y empaquetar los materiales correctos para la entrega de la versión. La biblioteca de *software* es un elemento clave en el cumplimiento de las tareas de liberación y entrega.

Cada vez que se finaliza un *sprint* se obtienen entregables de lo realizado y se tiene un conocimiento sobre qué versión de cada elemento de configuración está presente en los mismos. Hasta el momento solo se han tenido lanzamientos internos, pero se fijó una política en la que frente a instalaciones del *software* en clientes se debe tener una planilla actualizada sobre quien lo instalo, cuando, que versión y que cosas instalo.

A continuación se puede ver la última versión del plan de *release* adoptado, el cual contiene el periodo de tiempo de cada iteración y los *feature set* que se terminaron en dicha iteración con los puntos que implican los mismos.

Iteración	**Fecha de inicio**	**Fecha de fin**	***Feature Set***	**Puntos**
1	20/07/2015	17/08/2015	1	21
2	18/08/2015	30/08/2015	11	13
3	31/08/2015	13/09/2015	2	21
4	14/09/2015	27/09/2015	3	13
5	28/09/2015	11/10/2015	5, 4,12	19
6	12/10/2015	25/10/2015	7,6	13
7	26/10/2015	8/11/2015	8,18	21
8	9/11/2015	22/11/2015	14,13,9	24
9	23/11/2015	06/12/2015	15,16, 17,10	16
10	07/12/2015	20/12/2015	20	21

11	21/12/2015	03/01/2016	19,21	21

Tabla 9-1: plan de *release*

9.5 Herramientas de gestión de configuración de *software*

A la hora de hablar de herramientas de gestión de configuración de *software* se tomaron recaudos en cuanto a la posibilidad de acceder a las mismas desde múltiples plataformas, por la realidad de hoy en donde distintos sistemas operativos coexisten en la vida diaria de cada uno de los integrantes ya sean de escritorio o *mobile*. También se evaluaron herramientas las cuales alguno de los miembros tenga conocimiento del uso de la misma, mientras que por último se priorizaron herramientas de libre uso que no demanden un pago mensual por la utilización de sus servicios.

Esto nos llevó a la elección de las siguientes herramientas, para el control y mantenimiento de los elementos de configuración del *software*:

- Google Drive: para aquellos elementos del tipo documento o datos.
- Bitbucket: como repositorio del tipo GIT para el código fuente de todos los módulos de nuestro *software*.

10 Conclusiones y lecciones aprendidas

El proyecto GDR representó uno de los mayores desafíos y factores de crecimiento personal y profesional para los integrantes del equipo. La necesidad de auto gestionarse de forma eficaz y eficiente para satisfacer las necesidades académicas tanto como las comerciales fue uno de los mayores desafíos, de la cual el equipo obtuvo un enorme aprendizaje.

Retomando los objetivos planteados en la sección objetivos, se procederá a evaluar cada uno de ellos de forma autónoma.

10.1 Objetivos del proyecto

Se consolidó un equipo de trabajo capaz de resolver los problemas que se presentaran ante él, ya fuesen problemas técnicos, de negocio o de gestión.

El equipo logro obtener sólidos conocimientos en las tecnologías que se utilizaron para desarrollar el proyecto, también pudiendo obtener conocimientos de negocio gracias a una continua interacción con el cliente y enfocándose en satisfacer las necesidades desde su origen y no pensando una solución desde una tecnología o programa.

La conclusión del cliente fue que se evidenció un trabajo profesional y fue evolucionando de forma correcta, evolucionando siempre con cada *feedback* u oportunidad de mejora detectada.

Uno de los principales factores de que no se percibiese la calidad necesaria para que el proyecto entrase en una etapa de puesta en producción fue la falta de datos reales, los cuales no fueron provistos por el cliente según lo acordado.

10.2 Objetivos del producto

Finalmente no fue posible realizar una instalación en un cliente, debido a varios factores:

- Se tomó como cliente tipo uno sumamente exigente, lo cual fue un tanto ambicioso.
- La no recepción de datos a través de la interfaz planteada incurrió en un aumento de horas de pruebas y generación de datos de prueba para el equipo. Además que la falta de datos reales generó una disconformidad en el concepto de calidad del cliente a la hora de realizar pruebas de avance.
- Un cambio de alcance inesperado en los últimos meses del proyecto

Igualmente se considera que se generó un producto que genera información útil para los usuarios. Esto es posible de afirmar gracias a las pruebas de aceptación que fueron realizadas con usuarios finales, en las cuales los usuarios afirman que es un producto fácil de aprender a usar y sencillo.

Sin lugar a dudas aún existe lugar a mejora y así lo hicieron saber también los usuarios proporcionando su opinión sobre qué aspectos mejorarían aún más su experiencia. En concreto se podría simplificar el proceso de generación de un reporte previendo los criterios de agrupación u haciendo que el usuario los indique de una forma más amigable.

Según el cliente se concretó que el producto actualmente es de ágil aprendizaje ya que en todo momento tiene claro que acción está realizando, pero tiene lugar a mejora en cuanto a la primera interacción del usuario con el mismo. En concreto se podría definir más meta data sobre las columnas (esto para la definición de nuevos tipos de informes), o en base al orden que el usuario coloca los atributos en la pre visualización decidir qué datos se agrupan y cuáles no.

Otra gran fortaleza del producto que fue detectada durante su desarrollo es la facilidad de cruzar datos y detectar que datos e información es la que resulta de mayor interés para los usuarios. Esto permite afirmar que el producto puede ser utilizado como una herramienta de análisis de generación de cubos a un muy bajo costo, esto es una gran diferenciación ante las actuales herramientas de *reporting* que incurren en una gran carga monetaria de análisis y diseño.

10.3 Objetivos académicos

Los objetivos académicos fueron también alcanzados de forma exitosa, pudiendo los miembros del equipo adquirir conocimientos sobre nuevas tecnologías utilizadas cada vez con mayor frecuencia en el mercado laboral. De esta forma el equipo considera haber obtenido un enorme crecimiento como ingenieros de software.

Fue posible aplicar todos los conocimientos adquiridos durante la carrera, debiendo principalmente adaptarlos a las necesidades del proyecto y no viceversa. Se debió gestionar un proyecto de desarrollo de software con un foco inicial en investigación para formar una base sobre la cual trabajar, y luego optar por un *framework* de gestión que no era sumamente conocido por el equipo pero se adaptaba mejora a las características del proyecto.

10.4 Lecciones aprendidas

Durante el proyecto se cometieron tanto aciertos como errores, de los cuales fue necesario aprender y finalmente reflexionar sobre las decisiones que los provocaron.

Proceso de investigación

Contar con una etapa inicial de investigación enfocado en manejo de grandes volúmenes de información en un ambiente de herramientas de *reporting* fue uno de los mayores aciertos del equipo. Le dio al equipo una buena base de conocimiento, que posteriormente permitió argumentar decisiones tomadas o no tomadas con clientes o usuarios.

Haber realizado una investigación tanto de aspectos técnicos como teóricos permitió en etapas tempranas definir un conjunto de requerimientos sólido y una arquitectura inicial con fundamentos.

Incorporación de un *framework* de gestión

Adoptar un *framework* poco conocido para el equipo, como lo era FDD, fue todo un desafío, lo cual claramente tuvo un impacto negativo en etapas tempranas de la implementación del mismo. El equipo debió hacer un mayor énfasis en la detección de defectos y errores (tanto de proceso como de producto) en etapas tempranas para minimizar el impacto de la curva de aprendizaje y adaptación de un *framework* de gestión.

En los aspectos que se hizo más énfasis fue en controlar que se siguiera el flujo de trabajo definido por el *framework* previamente modificado para adaptarlo al proyecto en concreto.

Resolución de problemas

Durante el transcurso del proyecto el equipo debió enfrentarse a diversos problemas, algunos sencillos y otros complejos. Sin lugar a dudas estos últimos generaron inconformidad y frustración en el equipo, pero lo más enriquecedor siempre fue poder resolverlos de una forma ordenada y profesional.

Otra lección aprendida en este aspecto fue nunca sobre estimar un problema por más sencillo que parezca y siempre evaluar las características cuantitativas y cualitativas del mismo

Trato comercial

Lograr un trato fluido y basado en la confianza fue el objetivo desde un principio del equipo para con el cliente. En base a las numerosas reuniones llevadas a cabo con el cliente se fue consolidando un trato fluido, que junto con el compromiso demostrado y la continua transparencia ante la ocurrencia de inconvenientes, decantaron en una mejor detección de defectos, desconformidades o problemas.

Aún queda mucho por aprender en lo que respecta al trato comercial, pero creemos contar con una buena experiencia y haber encontrado una forma de trabajar basada en la confianza y seriedad.

Autogestión

Sin lugar a dudas uno de los aspectos en los que el equipo creció y aprendió en mayor medida. Gracias a la madurez que se fue obteniendo en este aspecto se logró seguir un proceso de desarrollo de forma estructurada y principalmente fue posible sobrellevar los cambios y/o desafíos que se fueron presentando de forma ordenada y profesional.

Debido a los distintos compromisos laborales y académicos del equipo durante el proyecto, fue clave definir la autogestión, una de las principales características del equipo.

11 Referencias

[1] «Interfaz de programación de aplicaciones - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones.

[2] «Lluvia de ideas - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Lluvia_de_ideas.

[3] «Cubo OLAP - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Cubo_OLAP.

[4] «Minería de datos - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Miner%C3%ADa_de_datos.

[5] «Data Warehouse - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Almac%C3%A9n_de_datos.

[6] «Software deployment - Wikipedia, the free encyclopedia,» [En línea]. Disponible: https://en.wikipedia.org/wiki/Software_deployment.

[7] «Proyect Stakeholder - Wikipedia, the free encyclopedia,» [En línea]. Disponible: https://en.wikipedia.org/wiki/Project_stakeholder.

[8] «JSON,» [En línea]. Disponible: http://www.json.org/.

[9] «Middleware - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Middleware.

[10] «Simple Object Access Protocol - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol.

[11] P. Bourque y R. E. Fairley, SWEBOK v3.0, New York City: IEEE, 2014.

[12] P. Coad, E. Lefebvre y J. DeLuca, Java Modeling in Color with UML, Prentice-Hall, 1999.

[13] S. Palmer y J. Felsing, A Practical Guide to Feature Driven Development, Prentice-Hall, 2002.

[14] R. S. Pressman, Software Engineering - A Practitioner's Approach, McGraw-Hill, 2005.

[15] K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland y D. Thomas, «Manifiesto por el Desarrollo Ágil de Software,» 2001. [En línea]. Disponible: http://www.agilemanifesto.org/iso/es/.

[16] «w3schools-json syntax,» [En línea]. Disponible: http://www.w3schools.com/json/json_syntax.asp.

[17] «Ingeniería inversa - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa.

[18] «Manual_M2C1U06_Adendum.pdf,» [En línea]. Disponible: http://is.ls.fi.upm.es/docencia/masterTI/ARS/docs/Manual_M2C1U06_Adendum.pdf.

[19] P. A. Len Silverston, The Data Model Resource Book Vol. 1, Wiley, 2001.

[20] «CloudHarmony,» [En línea]. Disponible: www.cloudharmony.com.

[21] «The groovy programming language - Style guide,» [En línea]. Disponible: http://groovy-lang.org/style-guide.html.

[22] «PSR-2: Coding Style Guide -PHP -FIG,» [En línea]. Disponible: http://www.php-fig.org/psr/psr-2/.

[23] «Principio de Pareto - Wikipedia, la enciclopedia libre,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Principio_de_Pareto.

[24] I. Sommerville, Software Engineering, Addison-Wesley, 2007.

[25] «IEEE Std 1044-2009 - IEEE Standard Classification for Software Anomalies,» IEEE Computer Society, [En línea]. Disponible: https://standards.ieee.org/findstds/standard/1044-2009.html.

[26] «The Apache Cassandra Project,» [En línea]. Disponible: http://cassandra.apache.org/.

[27] «The column-store pioneer | MonetDB,» [En línea]. Disponible: https://www.monetdb.org/Home.

[28] «Open Source Data Warehouse, Column Database Software, Improve SQL Performance - Infobright.org,» [En línea]. Disponible: http://www.infobright.org/.

[29] «Citus Data,» [En línea]. Disponible: https://github.com/citusdata/.

[30] «SQLite,» [En línea]. Disponible: http://www.sqlite.org/.

[31] «NoSQL - Wikipedia, the free encyclopedia,» [En línea]. Disponible: http://en.wikipedia.org/wiki/NoSQL.

12 Anexos

12.1 Requerimientos Funcionales

En el presente anexo se presenta evidencia fotográfica del proceso que se siguió para obtener el conjunto de *features* inicial del producto y la última versión del conjunto de *features* del producto.

12.1.1 Features

12.1.1.1 Configurar reportes

1.1	Seleccionar datos involucrados en el reporte
Muestra	Estructura de datos
En	Vista de creación de reporte (módulo: Web)
Para	Usuario pueda seleccionar qué datos desea ver en el reporte

Tabla 12-1: *feature* 1.1

1.2	Seleccionar tipo de reporte
Muestra	Diferentes opciones de reporte
En	Vista de creación de reporte (módulo: Web)
Para	Personalizar el reporte

Tabla 12-2: *feature* 1.2

1.3	Seleccionar filtros de la consulta
Muestra	Tipos de filtro
En	Vista de creación de reporte (módulo: Web)
Para	Indicar los criterios que deben cumplir los datos del reporte

Tabla 12-3: *feature* 1.3

1.4	Enviar solicitud de reporte
Envía	Datos de la consulta al módulo de traducción
En	IInterprete (interfaz de comunicación, módulo: Web)
Para	Obtener los datos del reporte

Tabla 12-4: *feature* 1.4

1.5	Mostrar datos del reporte
Muestra	Resultados obtenidos
En	Vista de reporte (módulo: Web)
Para	Usuario pueda ver el reporte

Tabla 12-5: *feature* 1.5

1.6	Seleccionar filtros en el reporte
Muestra	Tipos de filtro
En	Vista de reporte (módulo: Web)
Para	Filtrar datos

Tabla 12-6: *feature* 1.6

12.1.1.2 Traducir consulta

2.1	Interpretar consulta
Genera	Consulta a partir de los datos recibidos
En	Intérprete

Para	Poder ejecutar la consulta

Tabla 12-7: *feature* 2.1

2.2	Ejecutar consulta
Genera	Conjunto resultado
En	Manejador de consultas
Para	Obtener resultado

Tabla 12-8: *feature* 2.2

2.3	Procesar datos
Muestra	Conjunto resultado procesado
En	Intérprete
Para	Devolver resultado al módulo web

Tabla 12-9: *feature* 2.3

2.4	Devolver resultado
Genera	*Stream* de datos
En	IInterprete (interfaz de comunicación)
Para	Enviar resultados a la web

Tabla 12-10: *feature* 2.4

12.1.1.3 Guardar reporte localmente como archivo

3.1	Seleccionar formato
Genera	Instrucción de guardado

En	Vista de reporte (módulo: Web)
Para	Guardar reporte con un nombre que se indicará

Tabla 12-11: *feature* 3.1

3.2	Nombrar reporte
Genera	Instrucción de descarga
En	Vista de reporte (módulo: Web)
Para	Guardar el reporte en la máquina del usuario

Tabla 12-12: *feature* 3.2

12.1.1.4 Guardar reporte en el sistema

4.1	Guardar reporte en la base de datos
Modifica	Tabla de reportes en base de datos
En	Base de datos
Para	Registrar que se realizó la consulta

Tabla 12-13: *feature* 4.1

12.1.1.5 Reportes favoritos

5.1	Marcar reporte como favorito
Modifica	Tabla de reportes en base de datos
En	Vista de reporte (módulo: Web)
Para	Usuario pueda ver dicho reporte en otro momento

Tabla 12-14: *feature* 5.1

12.1.1.6 Modificar reportes

6.1	Seleccionar reporte a modificar*
Muestra	Datos del reporte
En	Vista de reportes favoritos (módulo: Web)
Para	Ver los posibles datos a modificar del mismo

Tabla 12-15: *feature* 6.1

*Los reportes se podrán elegir de entre los reportes favoritos del usuario.

6.2	Seleccionar valor del reporte a modificar
Habilita	Componente gráfico (campo de texto, combo box, etc.)
En	Vista de modificación de reporte (módulo: Web)
Para	Permitir al usuario indicar el valor deseado para la variable

Tabla 12-16: *feature* 6.2

6.3	Modificar valor
Modifica	Valor de la variable
En	Vista de modificación de reporte (módulo: Web)
Para	Generar el reporte con el valor deseado

Tabla 12-17: *feature* 6.3

6.4	Confirmar modificación
Guarda	Valor de variable
En	Vista de modificación de reporte (módulo: Web)

Para	Generar el reporte con el valor deseado

Tabla 12-18: *feature* 6.4

6.5	Modificar visualización
Habilita	Interfaz para modificar visualización
En	Vista de modificación de reporte (módulo: Web)
Para	Indicar la visualización preferida por el usuario

Tabla 12-19: *feature* 6.5

6.6	Seleccionar visualización
Guarda	Tipo de visualización
En	Vista de modificación de reporte (módulo: Web)
Para	Ver el reporte de la forma deseada por el usuario

Tabla 12-20: *feature* 6.6

12.1.1.7 Ver reporte favorito

7.1	Seleccionar reporte deseado
Muestra	Reportes favoritos del usuario
En	Vista de listado de reportes (módulo: Web)
Para	Usuario pueda seleccionar que reporte obtener

Tabla 12-21: *feature* 7.1

7.2	Confirmar datos del reporte
Envía	Datos de la consulta

En	IConsultas (interfaz de comunicación, módulo Web)
Para	Obtener los datos del reporte

Tabla 12-22: *feature* 7.2

12.1.1.8 Generación de reportes planificados

8.1	Seleccionar tipo de reporte
Determina	Tipo de reporte a generar
En	Vista de reporte planificado (módulo: Web)
Para	Mostrar el tipo de reporte deseado por el usuario

Tabla 12-23: *feature* 8.1

8.2	Seleccionar datos involucrados
Muestra	Estructura de datos
En	Vista de reporte planificado (módulo: Web)
Para	Armar consulta

Tabla 12-24: *feature* 8.2

8.3	Seleccionar fecha
Determina	Fecha inicial en la que se genera el reporte
En	Vista de reporte planificado (módulo: Web)
Para	Determinar el primer momento en el que se generará el reporte

Tabla 12-25: *feature* 8.3

8.4	Seleccionar frecuencia

Determina	Intervalo de tiempo entre cada reporte
En	Vista de reporte planificado (módulo: Web)
Para	Determinar cada cuantos días se generará el reporte

Tabla 12-26: *feature* 8.4

8.5	Guardar planificación
Modifica	Tabla de reportes planificados
En	Base de datos
Para	Determinar el primer momento en el que se generará el reporte

Tabla 12-27: *feature* 8.5

12.1.1.9 Manejo de usuarios

9.1.1	Registrarse
Muestra	Formulario para registrarse en el sistema
En	Vista de registro de usuarios (módulo: Web)
Para	Poder comenzar a usar el sistema

Tabla 12-28: *feature* 9.1.1

9.1.2	Indicar datos del usuarios
Modifica	Valores de los campos del formulario
En	Vista de registro de usuarios (módulo: Web)
Para	Determinar los valores de las variables del usuario

Tabla 12-29: *feature* 9.1.2

9.1.3	Guardar datos
Modifica	Tabla de usuarios
En	Base de datos
Para	Almacenar los datos especificados en el sistema y crear el nuevo usuario

Tabla 12-30: *feature* 9.1.3

9.2.1	Modificar usuario
Muestra	Formulario para editar datos de usuario
En	Vista de modificar usuarios (módulo: Web)
Para	Poder modificar datos de un usuario

Tabla 12-31: *feature* 9.2.1

9.2.2	Indicar datos del usuarios
Modifica	Valores de los campos del formulario
En	Vista de registro de usuarios (módulo: Web)
Para	Determinar los valores de las variables del usuario

Tabla 12-32: *feature* 9.2.2

9.2.3	Guardar cambios
Modifica	Tabla de usuarios
En	Base de datos
Para	Almacenar los datos especificados en el sistema y crear el nuevo usuario

Tabla 12-33: *feature* 9.2.3

9.3	Eliminar usuario
Modifica	Tabla de usuarios (estado del usuario, activo o inactivo)
En	Base de datos
Para	No tener una cuenta en el sistema

Tabla 12-34: *feature* 9.3

12.1.1.10 Privilegios

10.1.1	Asignar rol a usuario
Muestra	Roles del sistema
En	Vista de modificar usuarios (módulo: Web)
Para	Agregar o quitar privilegios a un usuario

Tabla 12-35: *feature* 10.1.1

10.1.2	Guardar cambios
Modifica	Tabla de usuarios
En	Base de datos
Para	Guardar los cambios especificados por el usuario

Tabla 12-36: *feature* 10.1.2

10.2.1	Asignar privilegios a un rol
Muestra	Privilegios del sistema
En	Vista de modificar rol (módulo: Web)
Para	Agregar o quitar privilegios a un rol

Tabla 12-37: *feature* 10.2.1

10.2.2	Guardar cambios
Muestra	Tabla de roles
En	Base de datos
Para	Guardar los privilegios de un rol

Tabla 12-38: *feature* 10.2.2

12.1.1.11 Recibir datos de PSIG

11.1	Recepción de información
Recibe	Información sobre modificación en base de datos de PSIG
En	LectorGenérico (Transformador)
Para	Procesar los datos y posteriormente replicarlos en nuestra BD

Tabla 12-39: *feature* 11.1

12.1.1.12 Procesar e insertar datos en base de datos en columnas

12.1	Procesamiento de datos
Procesa	Datos recibidos
En	LectorGenérico (Transformador)
Para	Facilitar el duplicado en nuestra BD

Tabla 12-40: *feature* 12.1

12.2	Inserción
Modifica	Tabla/s
En	Base de datos

Para	Nuestra BD tenga la misma información que PSIG

Tabla 12-41: *feature* 12.2

12.1.1.13 Sesión

13.1	Inicio de sesión
Permite	Acceso al sistema
En	Vista de inicio de sesión (módulo Web)
Para	Usuario use el sistema

Tabla 12-42: *feature* 13.1

13.2	Cierre de sesión
Quita	Acceso al sistema
En	Vista de cierre de sesión (módulo Web)
Para	Nadie pueda usar el sistema con su nombre de usuario

Tabla 12-43: *feature* 13.2

12.1.1.14 Solicitud de permiso

14.1	Solicitar permiso de acceso
Genera	Notificación al administrador para que otorgue permisos si lo considera pertinente
En	Vista de inicio (módulo Web)
Para	Usuario pueda usar las funcionalidades del sistema

Tabla 12-44: *feature* 14.1

14.2	Otorgar permisos

Permite	Acceso a las funcionalidades del sistema a un usuario
En	Vista de permisos (módulo Web)
Para	Atender las solicitudes de permisos

Tabla 12-45: *feature* 14.2

12.1.1.15 Enviar reporte por mail

15.1	Envío por mail
Envía	Correo electrónico al receptor con el reporte elaborado por el emisor
En	Vista de reporte (módulo Web)
Para	Comunicar a un usuario acerca del resultado de un reporte

Tabla 12-46: *feature* 15.1

12.1.1.16 Generar reporte a partir de otro

16.1	Seleccionar reporte
Muestra	Datos del reporte
En	Vista de configuración del reporte (módulo Web)
Para	Modificar los datos del mismo a gusto del usuario

Tabla 12-47: *feature* 16.1

La ejecución de éste *feature* continua en 1.1.

12.1.1.17 Log de actividad

19.1	Log de actividad
Modifica	Tabla de log

En	Base de datos
Para	Registrar que usuario realizó qué actividad

Tabla 12-48: *feature* 19.1

12.1.1.18 Ejecución de consulta

18.1	Recibir consulta
Obtiene	Consulta SQL
En	Manejador de consultas
Para	Saber que consulta ejecutar sobre la base de datos

Tabla 12-49: *feature* 18.1

18.2	Log de actividad
Selecciona	Datos de la base de datos en columnas
En	Base de datos
Para	Obtener los datos requeridos por el usuario

Tabla 12-50: *feature* 18.2

18.3	Retornar datos
Envía	Datos seleccionados
En	Manejador de consultas
Para	Mostrar los datos al usuario

Tabla 12-51: *feature* 18.3

12.1.1.19 Cambiar base en columnas a Infobright

19.1	Traducir scripts de MonetDB a Infobright
Obtiene	Scripts de la base en Infobright
En	Base de Datos
Para	Poder crear la base en Infobright

Tabla 12-52: *feature* 19.1

19.2	Integrar Infobright al Proyecto
Obtiene	Integración con Infobright
En	Modulo Web
Para	Poder manejar la base de Infobright

Tabla 12-53: *feature* 19.2

12.1.1.20 Cambiar interfaz gráfica del usuario

20.1	Relevar mejoras en base al *feedback* de usuarios y cliente
Obtiene	Nuevos *mockups* del diseño de la interfaz web
En	Modulo Web
Para	Poder implementar dicho diseño en la web

Tabla 12-54: *feature* 20.1

20.2	Construir nuevo diseño de la interfaz web
Obtiene	Nueva página web
En	Modulo Web

Para	Ofrecer interfaz más intuitiva y agradable al usuario final

Tabla 12-55: *feature* 20.2

12.1.1.21 Compartir reporte

21.1	Permitir ver reporte compartido
Obtiene	Nueva interfaz gráfica en la web, en la parte de se ubican mis reportes
En	Modulo Web
Para	Permitir que los usuarios se compartan reportes

Tabla 12-56: *feature* 21.1

21.2	Dar permisos a usuarios que se compartió el reporte
Obtiene	Diferentes permisos para ejercer acciones sobre los reportes
En	Modulo Web
Para	Para solo permitirle hacer sobre el reporte lo que tiene permitido

Tabla 12-57: *feature* 21.2

12.1.2 Proceso de obtención

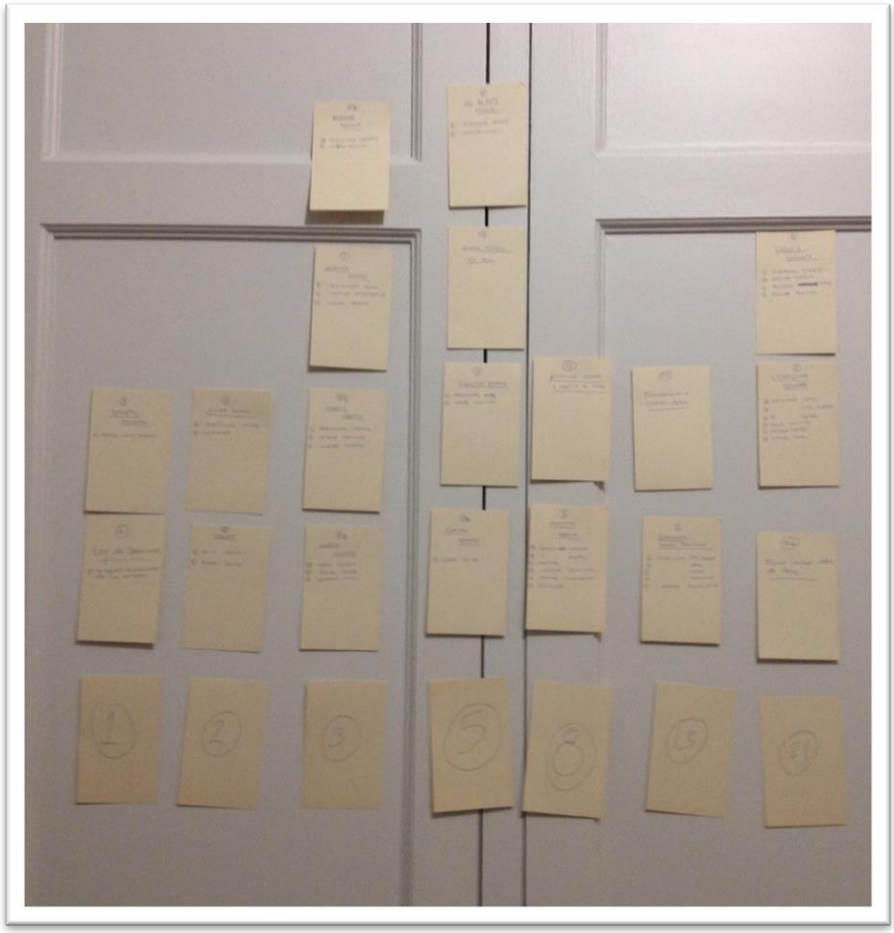

Figura 12-1: proceso de obtención de *features*

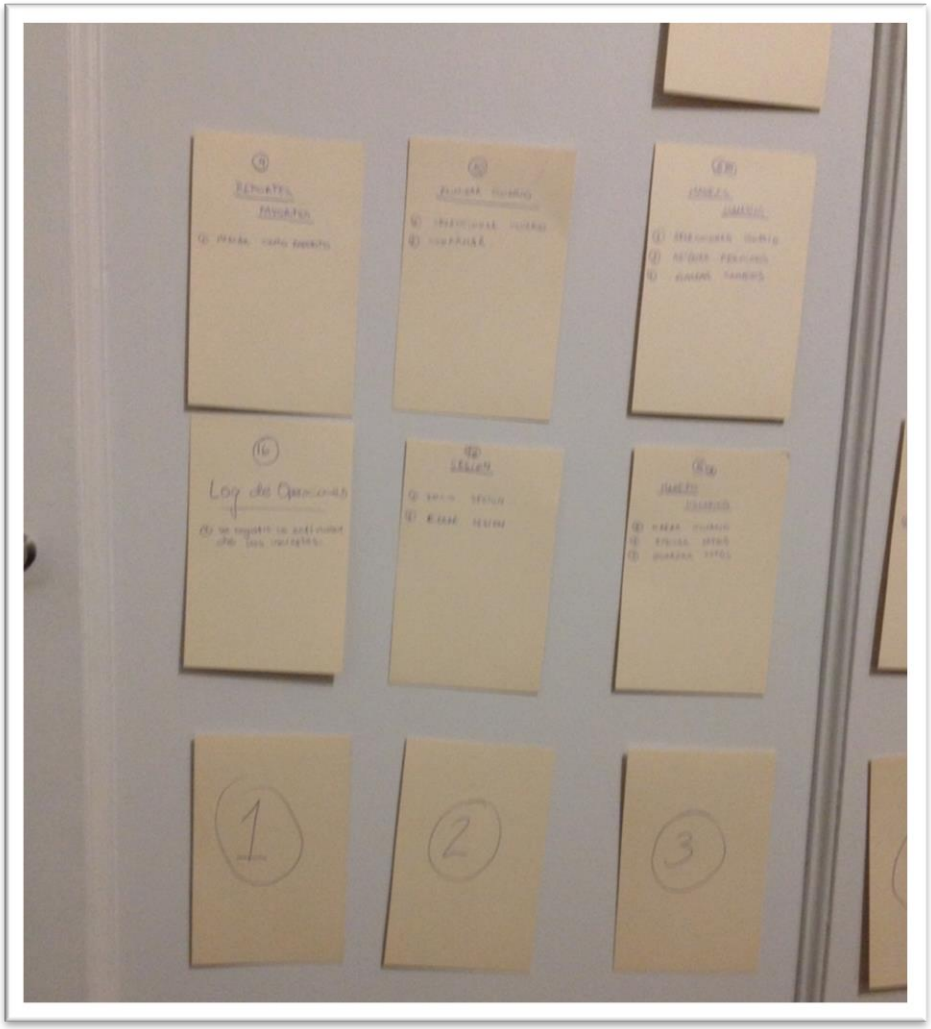

Figura 12-2: proceso de obtención de *features*

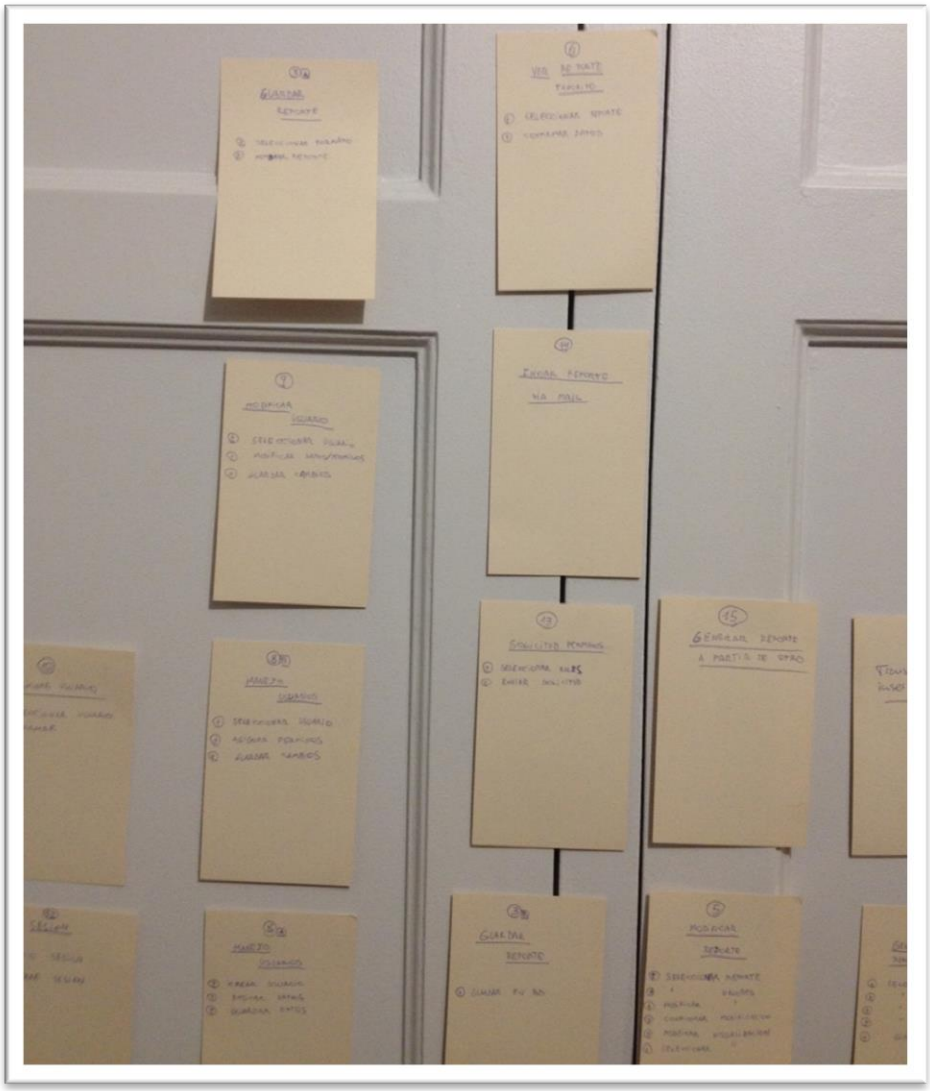

Figura 12-3: proceso de obtención de *features*

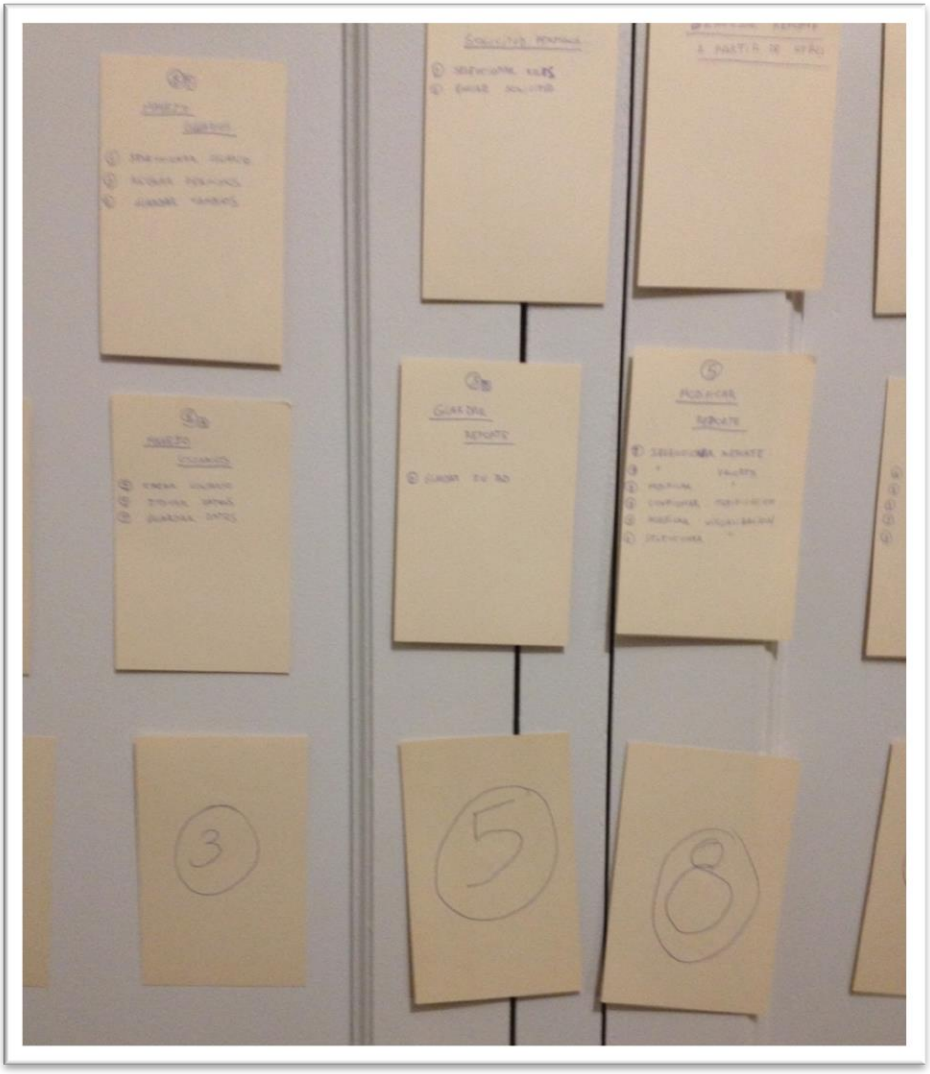

Figura 12-4: proceso de obtención de *features*

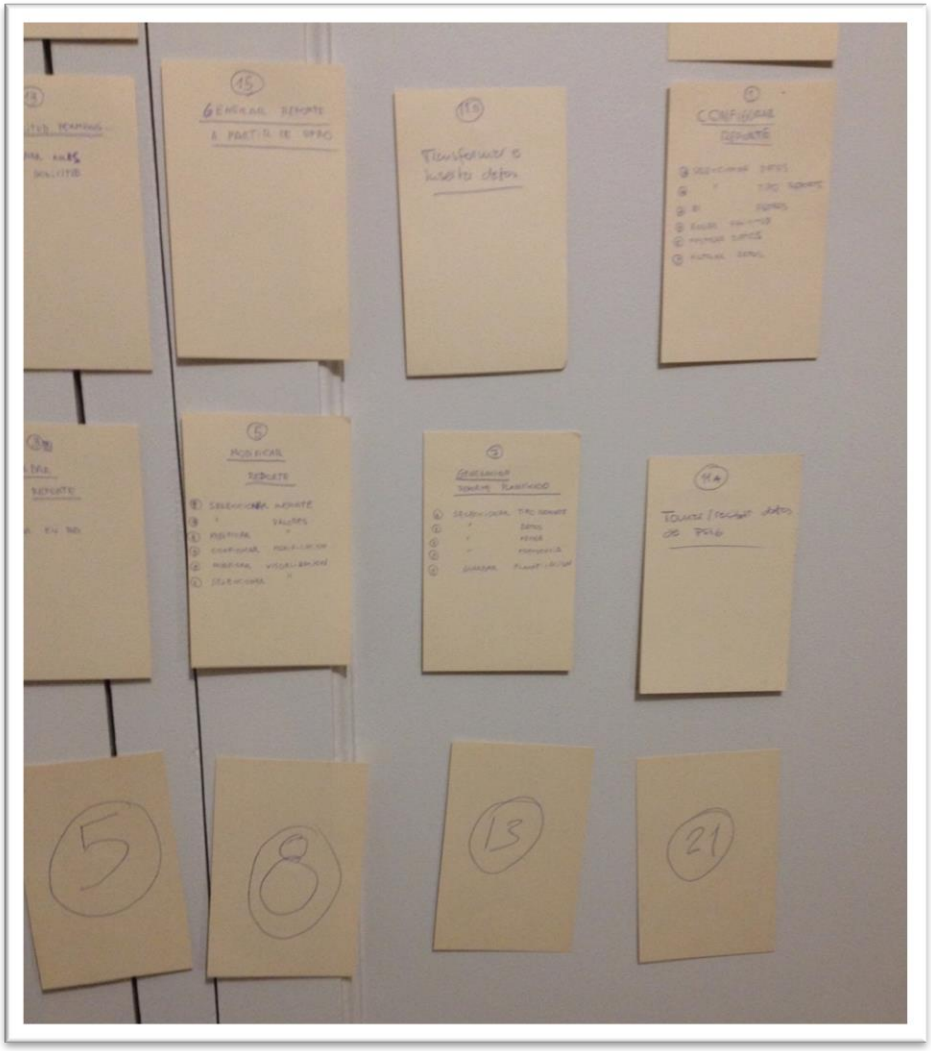

Figura 12-5: proceso de obtención de *features*

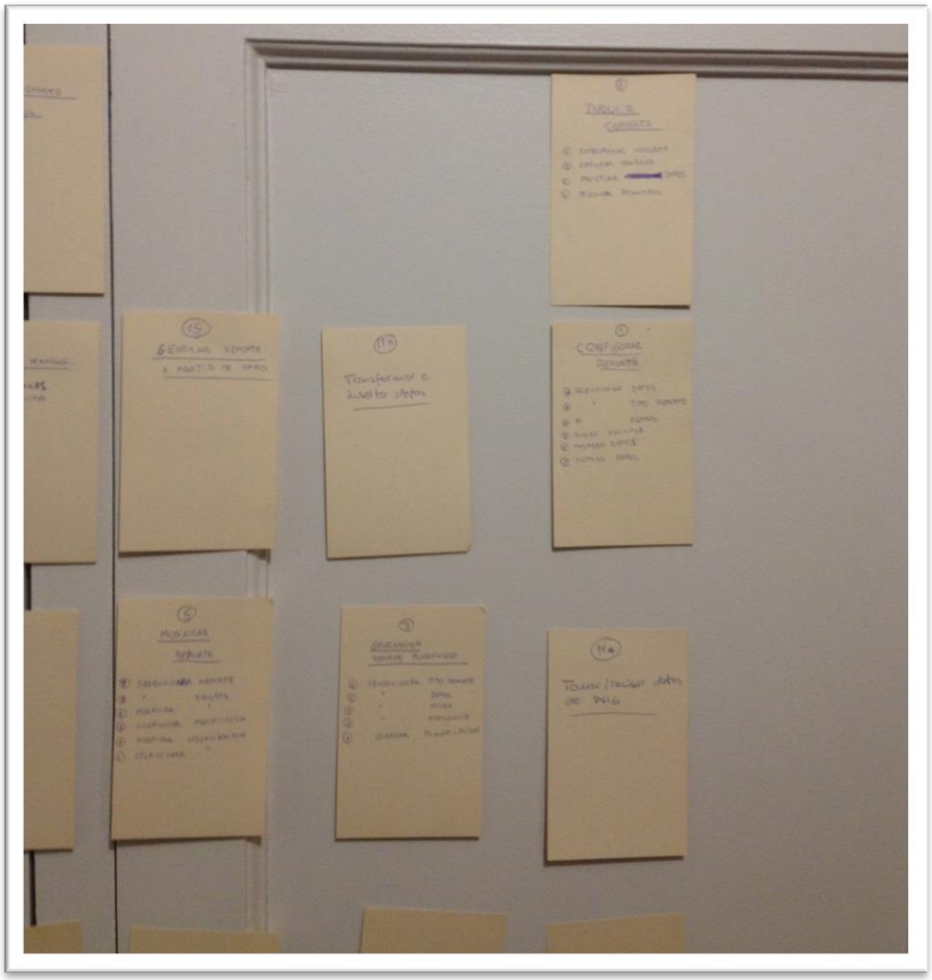

Figura 12-6: proceso de obtención de *features*

12.2 Modelos Conceptuales

12.2.1 Ventas

Ventas

{"venta":{

"cliente":{

"nombre":"Motociclo",

"tipoDoc":"RUT",

"numDoc":1321654651,

"ramo":"De flores",

```
                "infoAdicional":{
                        "edad":44,
                        "sexo":"Masculino"}
                },
"vendedor":{
            "nombre": "Jorge",
        "tipoDoc": "Cedula",
        "numDoc": 43213213,
            "infoAdicional": {
        "edad": 44,
        "sexo": "Masculino"}
    },
"documento": {
         "descripcion": "venta al por mayor",
    "fechaVencimiento": "2015-00-00",
    "tipoDePago": "credito",
    "tipoDoc": "RUT",
    "numDoc": 1321654651,
    "infoAdicional": {
          "ivaincluido": 1,
           "total": 450}
```

```
        },
      "departamento": {
        "nombre": "nombreDep",
        "codigoDepartamento": 1234,
        "algoMas" : "que consideren necesario"
      },
  "lineas": [
        {"codigo": "AX45", "descripcion": "item1","monto": 1321,"moneda": "dolar","cantidad": 1, "familia": {"nombre": "ACC GENE","descripcion": "Accesorios Genericos"}},
        {"codigo": "AX40", "descripcion": "item2","monto": 12120,"moneda": "peso uruguayo","cantidad": 1, "familia": {
"nombre": "ACC GENEBOL",
"descripcion": "Bolsitos"
}},
{"codigo": "AX49", "descripcion": "item3","monto": 131,"moneda": "euro","cantidad": 1, "familia": {
"nombre": "EQUI",
"descripcion": "Equipos"
}}
        ],
"local": {
"sucursal": 1,
```

"nombre": "Artigas",

"direccion": "Arocena 1234",

"infoAdicional": {

"casacentral": 1,

"barrio": "Carrasco"

}

}

}}

12.2.2 Compras

Compras

```
{"compra":{
        "empresa":{
                "nombre":"Supermercado Geant",
                "rut" : 213458920015,
                "razonSocial" : "Odelar SA",
                "direccion" : "Av a la playa 214",
                "telefono" : 29002289,
                "vendedor" : "Jorge Perez",
                "informacionAdicional" : {
                        "info" : "abcd"
                }
        },
        "comprador" : {
                "nombre" : "Esteban Quito",
                "tipoDoc" : "Cedula",
                "numDoc" : 47425655,
                "cargo" : "Asistente contable",
                "edad" : 44,
                "sexo" : "M"
```

```
	},
	"autorizante" : {
		"nombre" : "Yoni Compro",
		"tipoDoc" : "Cedula",
		"numDoc" : 44444444,
		"cargo" : "Gerente de compras"
	},
	"documento" : {
		"descripcion" : "Compra",
		"fechaVencimiento" : "2015-08-27",
		"tipoDePago" : "Contado",
		"nroFiscal" : "A1140169",
		"moneda" : "UYU",
		"total" : 1450,
		"ivaincluido" : 1,
		"informacionAdicional" : {
			"info" : "abcd"
		}
	},
	"articulos" : [
		{"nombre": "Jabon", "descripcion": "Jabón de glicerina para baño",
"monto": 1321, "cantidad": 10 },
		{"nombre": "Detergente", "descripcion": "Detergente para limpieza", "monto"
: 1321, "cantidad": 2 },
		{"nombre": "Papel higienico", "descripcion": "Detergente para limpieza",
"monto" : 1321, "cantidad": 2 }
		]
}}
```


12.3 Pruebas de concepto de base de datos

Primero que nada se realizó una revisión semiestructurada de bibliografía referente a consulta de grandes volúmenes de información de forma ágil.

A continuación se presentan los resultados de la búsqueda semi estructurada.

12.3.1 SQL

12.3.1.1 Orientado a filas

MySql y Postgresql

Ventajas

- Muy conocidas
- Fáciles de usar

Desventajas

- No escalan bien con altos volúmenes de datos por defecto

Usos

- BD más estudiada y conocida en el mercado actualmente

12.3.1.2 Orientado a columnas

- Cassandra (optimizada para escritura) [26]
- Monet [27]
- Infobright [28]
- CitusFDW [29]

Ventajas

- Pensada para lectura rápida
- Algoritmo de compresión que permite hacer las consultas más rápidas
- Soporta inserciones en lote

Desventajas

- No responde bien a inserciones ad-hoc
- Se necesita cierto requerimiento básico de hardware para sacarle provecho

Requerimientos particulares de hardware para optimización

- 32 GB ram

- 2 TB disco

Usos

- Son la mejor opción para aproximarse desde las bases de datos relacionales.
- Inicialmente se puede generar una única tabla gigante con todos los datos y consultar de ella. Luego se puede tratar de refinar.

12.3.1.3 Embebidas

SQLite (sumamente viable si existen restricciones de costos en infraestructura) [30], ScimoreDB y BerkeleyDB

Ventajas

- Muy ligeras
- Altamente portables

Desventajas

- Volátiles
- Acotadas en espacio y procesamiento
- No están pensadas para uso concurrente

Usos

- Para correr junto con la aplicación. Tener en cuenta que si se hace una aplicación estas correrán en el browser del cliente.

12.3.2 NoSql

Data Model	***Performance***	***Scalability***	***Flexibility***	***Complexity***	***Functionality***
Relational Database	*variable*	*variable*	*low*	*moderate*	*relational algebra*
Key–Value Store	*high*	*high*	*high*	*none*	*variable (none)*
Graph Database	*variable*	*variable*	*high*	*high*	*graph theory*
Document-Oriented Store	*high*	*variable (high)*	*high*	*low*	*variable (low)*
Column-Oriented Store	*high*	*high*	*moderate*	*low*	*minimal*

Tabla 12-58: comparación de distintas tecnologías de base de datos [31]

12.3.2.1 Clave Valor

Redis, Memcached y BigTable.

Ventajas

- Estructura sencilla
- Excelentes tiempos de respuesta

Desventajas

- Requieren requerimientos mínimos de hardware para responder en los mejores tiempos
- Requerimientos particulares de hardware para optimización
- Mismos que para las BD orientadas a columnas

12.3.2.2 Documentales

MongoDB y ElasticSearch

Ventajas

- Replicación automática en los distintos nodos existentes
- Al no tener una estructura definida resultan bastantes flexibles

Desventajas

- Si existe un solo nodo no se les puede sacar el mejor provecho
- Al no tener una estructura definida es necesario tener cuidado con las versiones de los documentos

Requerimientos particulares de hardware para optimización

- Mismo que para la BD orientadas a columnas

Usos

Para poder generar informes con los últimos datos (lo no procesado todavía en nuestra base desde la base del cliente) mantenemos en estas bases los datos y directamente con una estructura consultable por el usuario.

12.3.3 Extras

Logstash (saca los logs de apache y se los manda a elasticsearch). Pero esta opción agrega mucha complejidad y se depende un muy prolijo manejo de *logs* lo cual no es muy recomendable que un sistema dependa de *logs*.

En base a los resultados obtenidos y asesoramiento con un experto de dominio el equipo decidió que la opción más segura y viable era trabajar con las bases de datos en columnas. Estas cuentan con un gran soporte en el mercado a la hora de optimizar las operaciones de lectura, y se puedo confirmar que ya existen empresas de gran porte que utilizan estas tecnologías lo cual proveyó mayor tranquilidad a la hora de tomar la decisión.

Con los conocimientos que el equipo poseía sobre herramientas de *reporting* y los resultados de la búsqueda semiestructurada se planteó como alternativa trabajar en un entorno *cloud* (donde la rapidez de consulta sería obtenida gracias al procesamiento en paralelo y replicación de los datos) o enfocarse en una tecnología sumamente performante de forma *inhouse*.

Analizando la viabilidad de cada opción con el cliente se descartó la opción *cloud* a pedido del mismo y se decidió por la opción *inhouse*.

12.4 Pruebas de usabilidad

***Mockups* para primera versión**

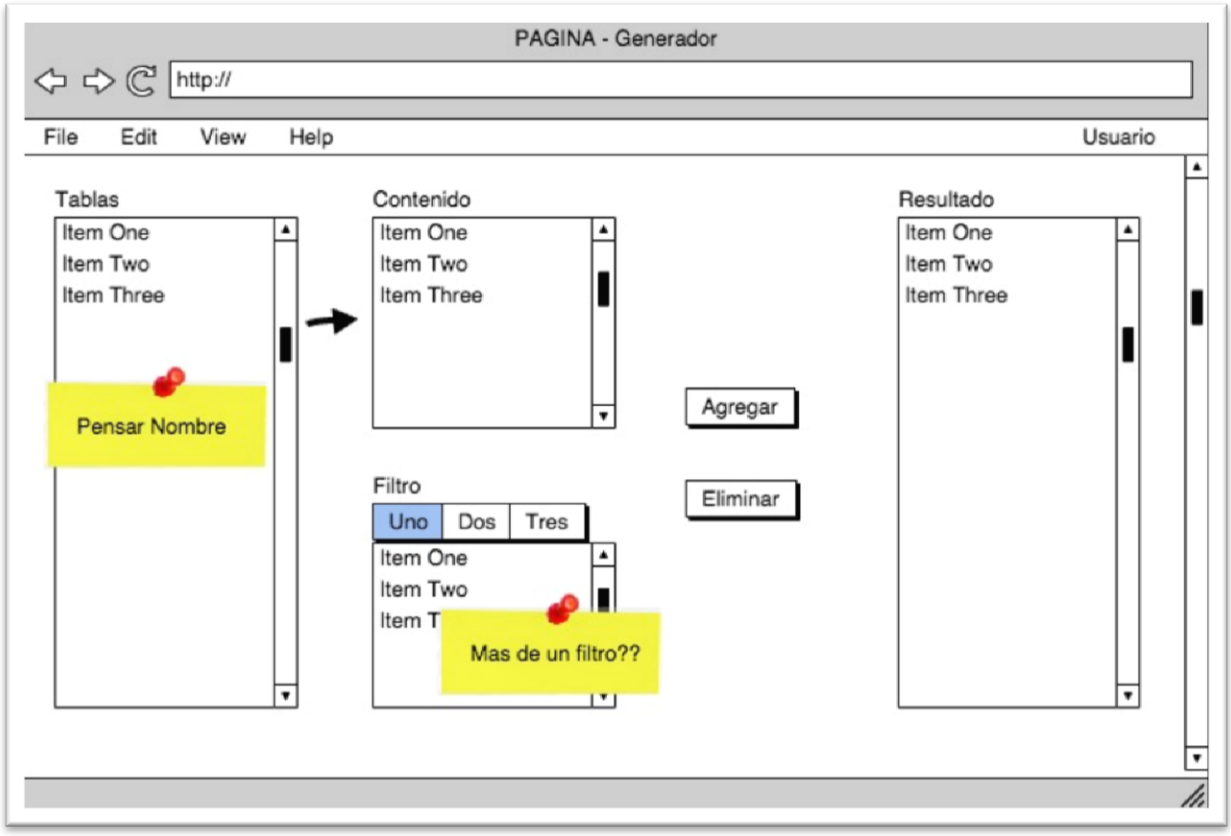

Figura 12-7: primera versión de *mockup* de UI

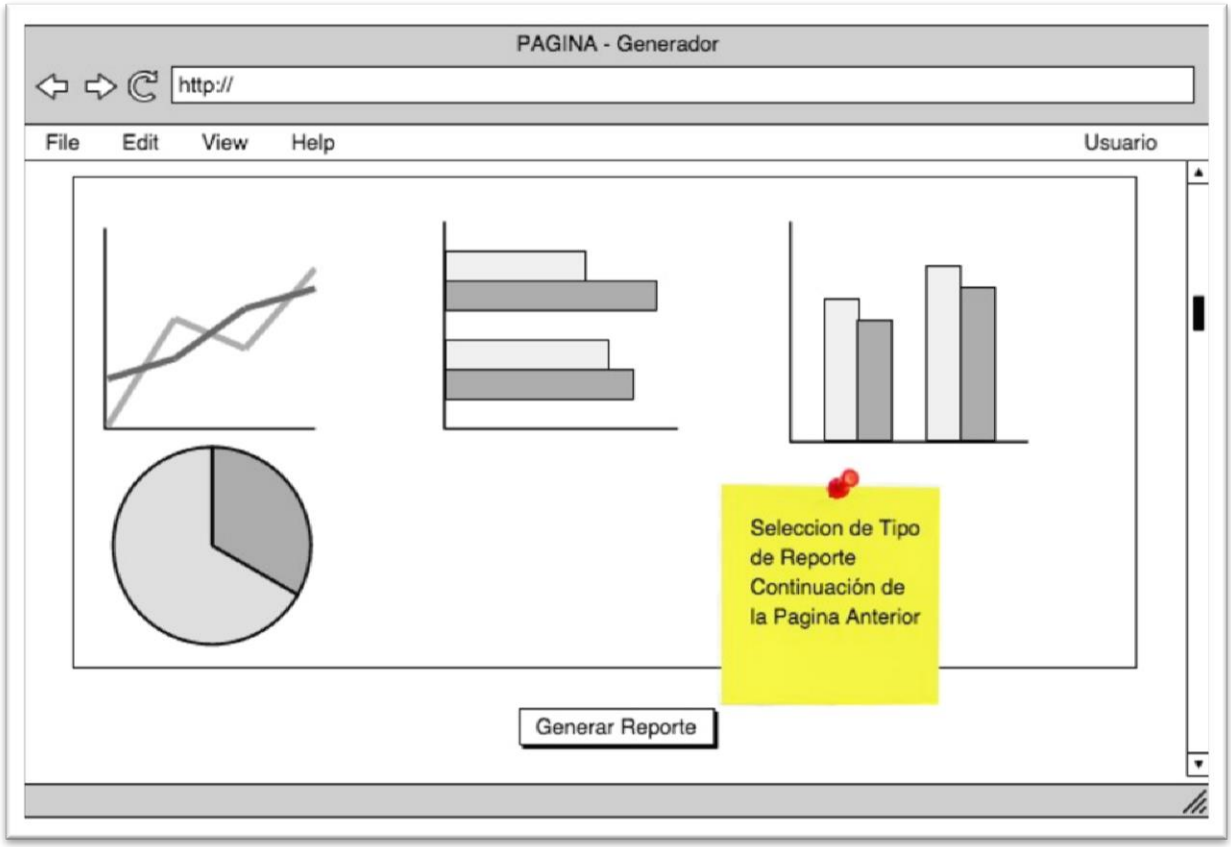

Figura 12-8: primera versión de *mockup* de UI

Luego los *mockups* evolucionaron:

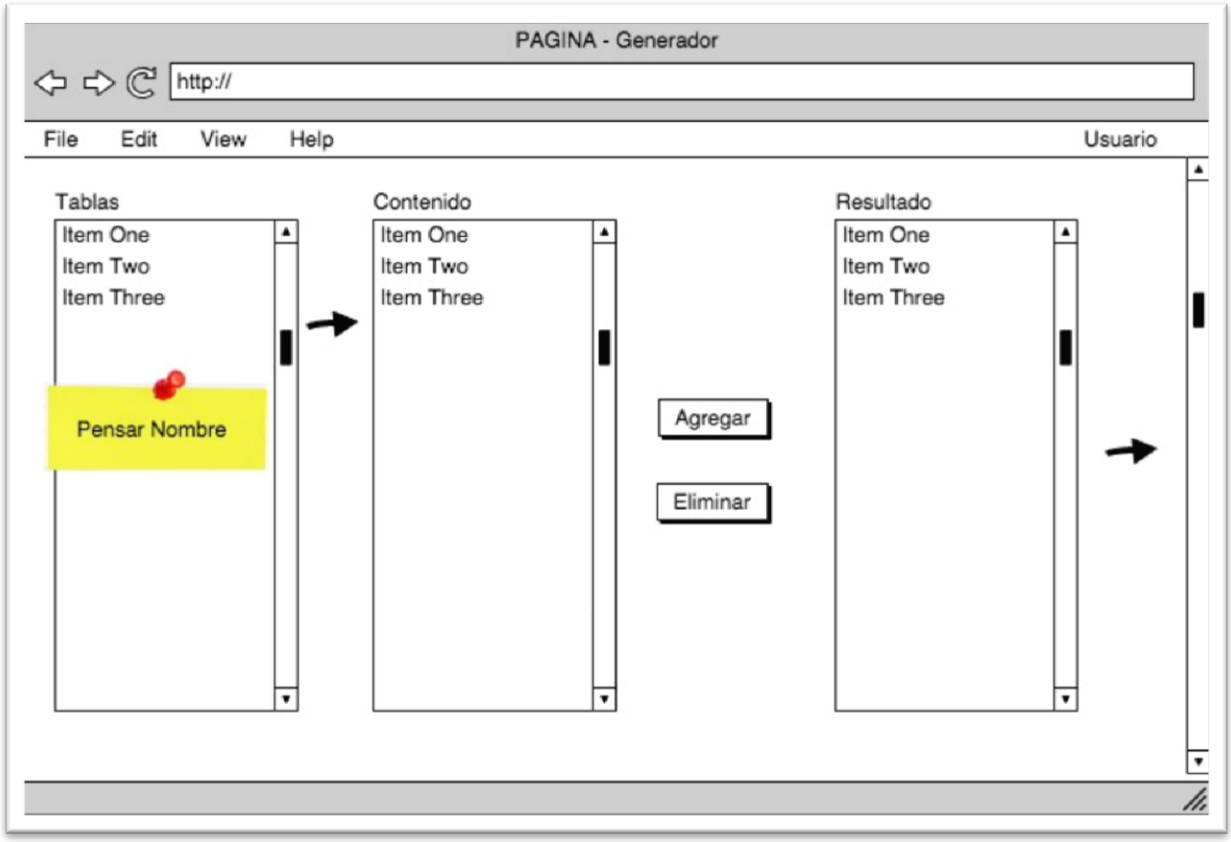

Figura 12-9: segunda versión de *mockup* de UI

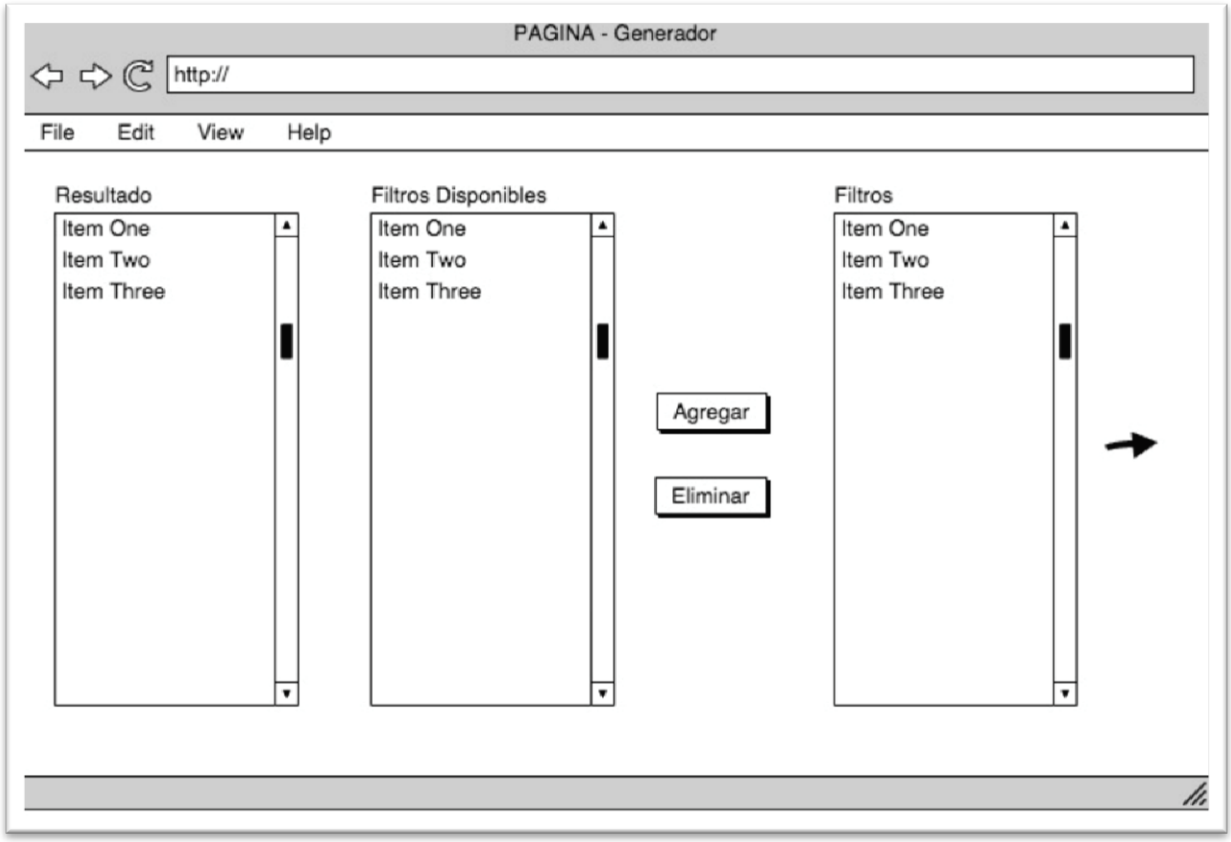

Figura 12-10: segunda versión de *mockup* de UI

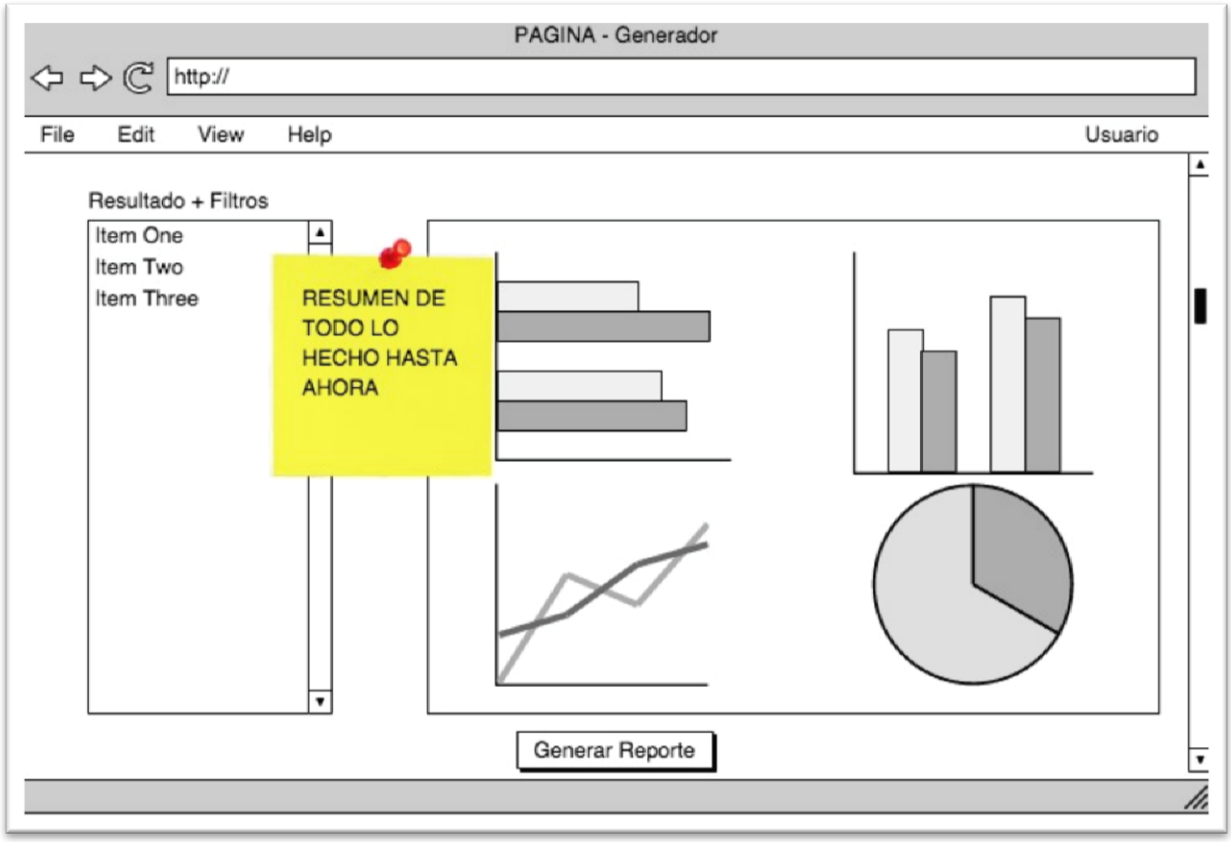

Figura 12-11: segunda versión de *mockup* de UI

Primera versión funcional

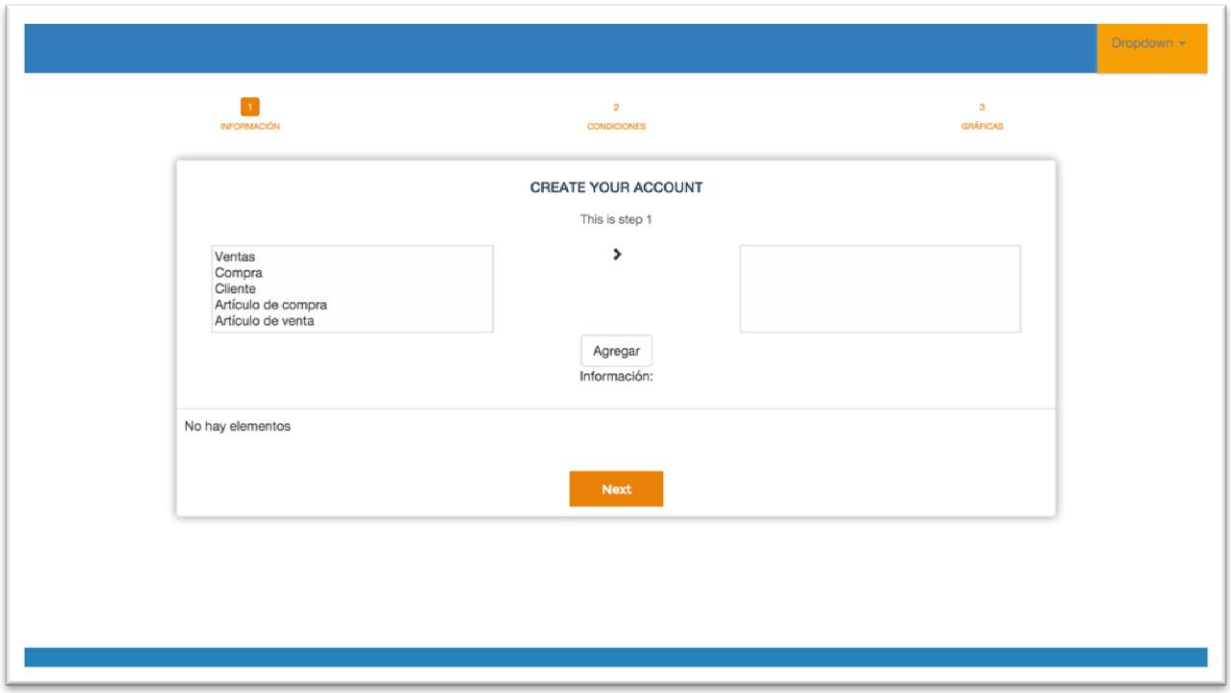

Figura 12-12: primer paso de la primera versión funcional

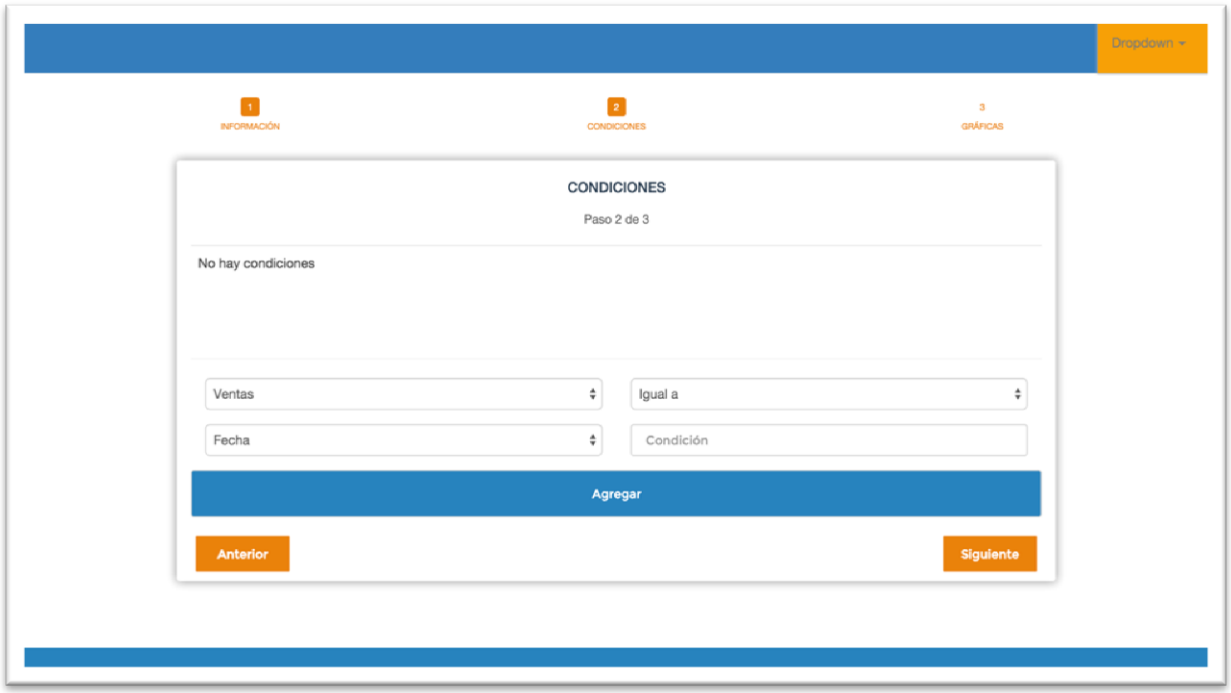

Figura 12-13: segundo paso de la primera versión funcional

Figura 12-14: tercer paso de la primera versión funcional

Mockup de evolución a nueva versión

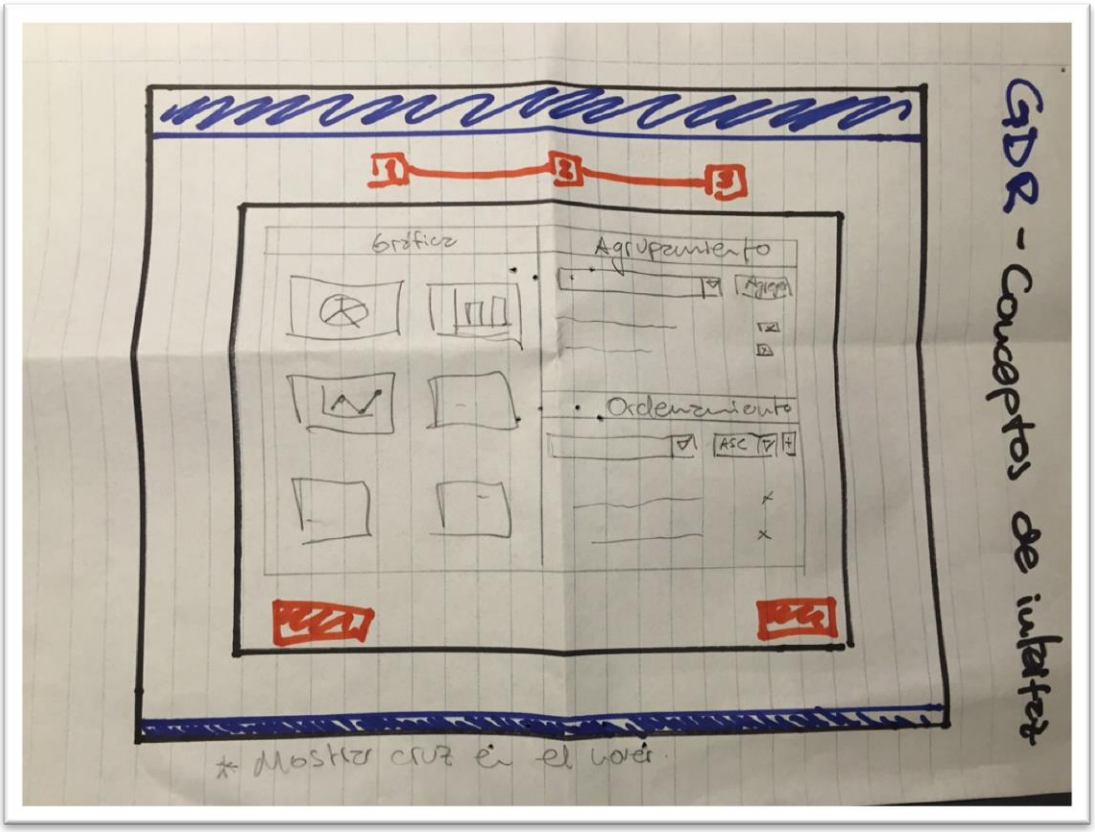

Figura 12-15: *mockup* de la evolución de la primera versión funcional

Segunda versión funcional

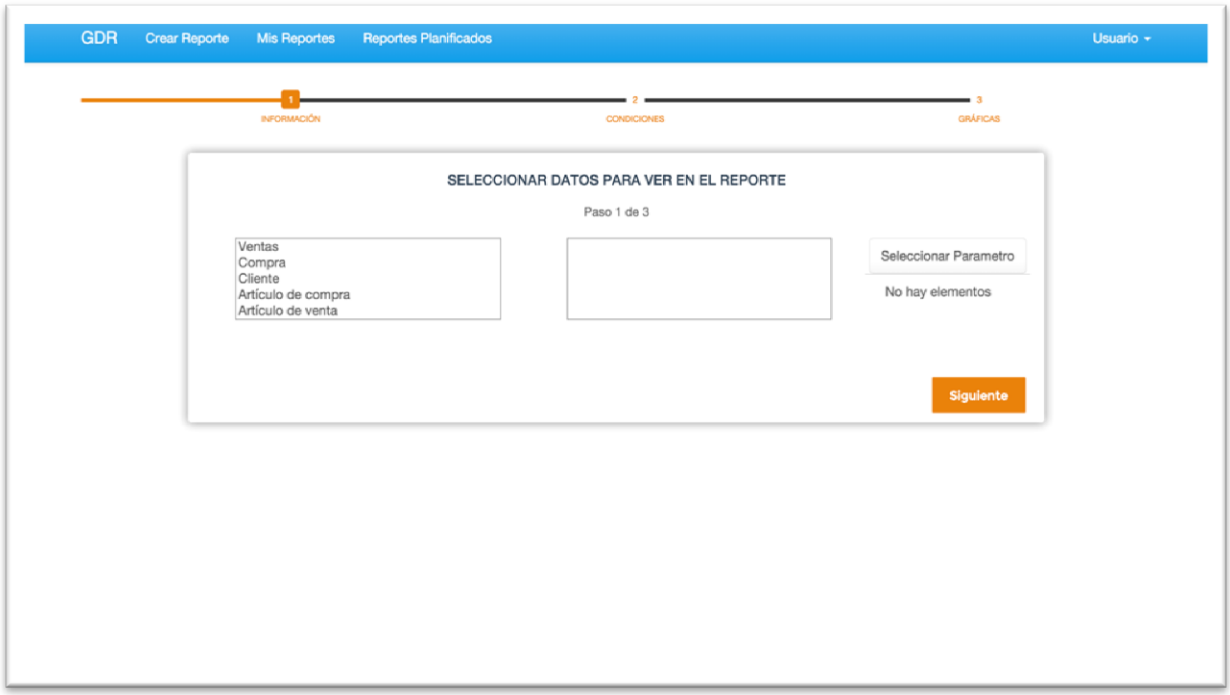

Figura 12-16: primer paso de la segunda versión funcional

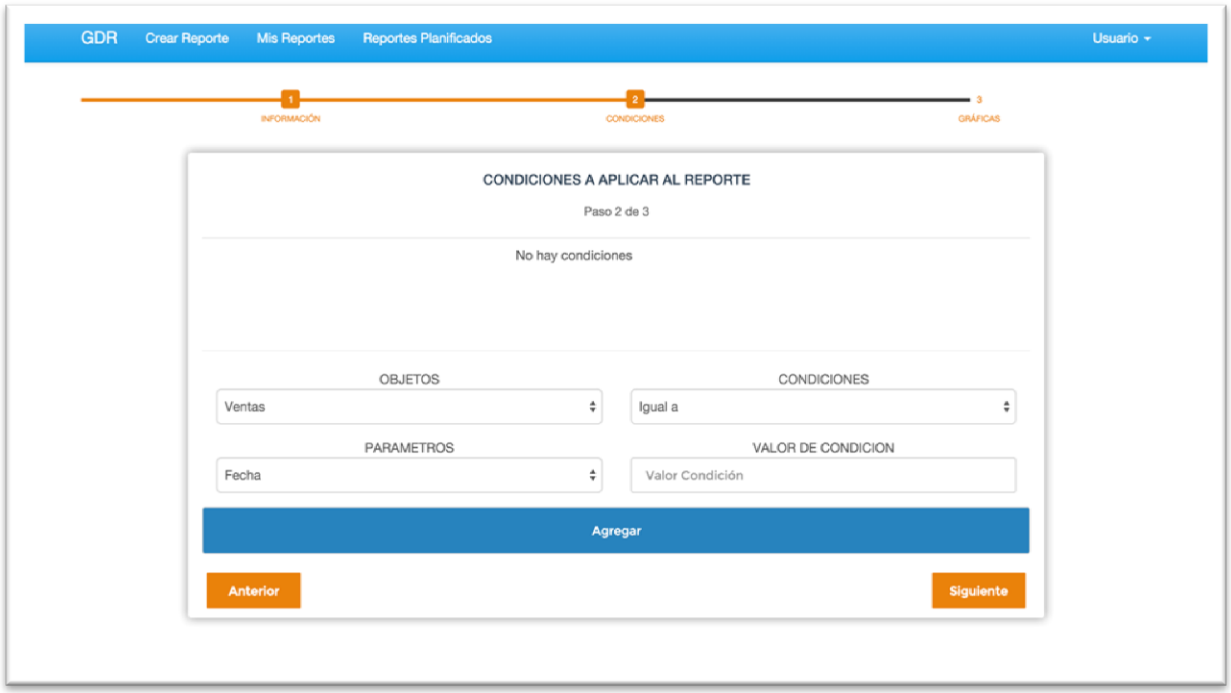

Figura 12-17: segundo paso de la segunda versión funcional

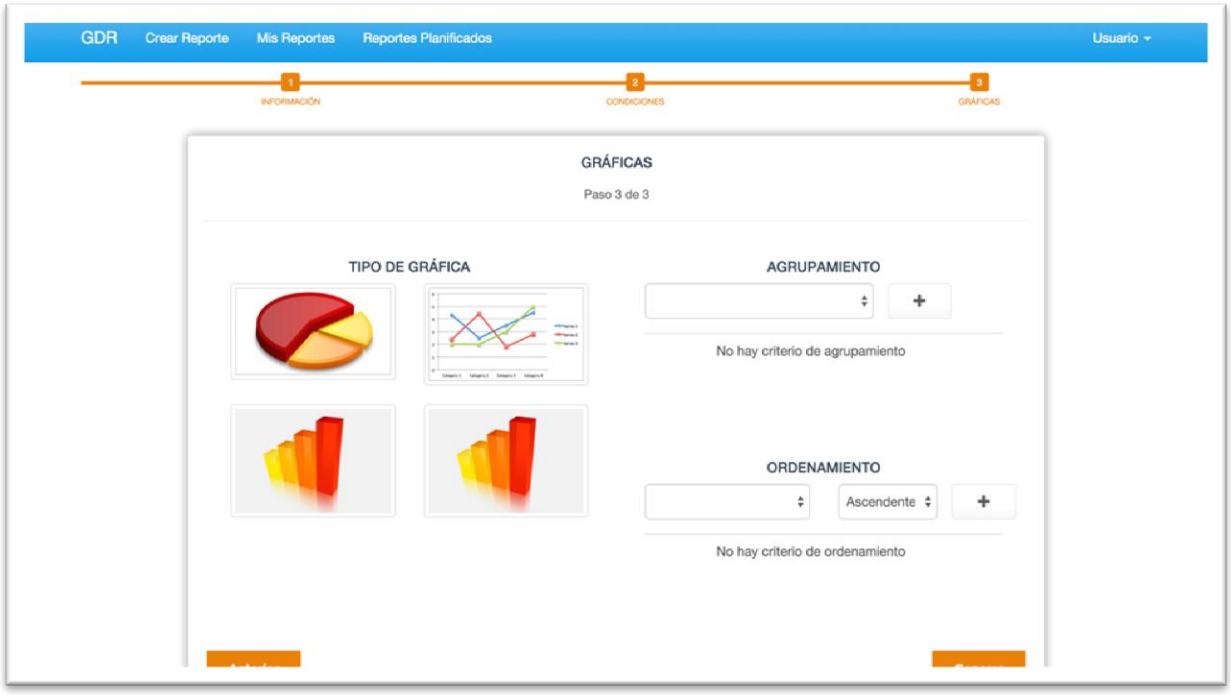

Figura 12-18: tercer paso de la segunda versión funcional

12.5 Arquitectura del Software

Se evidencian en el presente anexo las principales vistas y componentes que describen el sistema GDR en su totalidad.

12.5.1 Vistas estáticas

12.5.1.1 Despliegue

Existen variantes en cuanto a la disposición de los componentes en los distintos nodos, pudiendo cada componente ser ubicado en un nodo aislado. También resultaría posible ubicar toda la solución en un único nodo, pero no es la topología ideal que se plantea. El equipo creyó que con la topología planteada se puede proponer un servidor con las características mínimas necesarias para un adecuado funcionamiento de cada elemento del sistema.

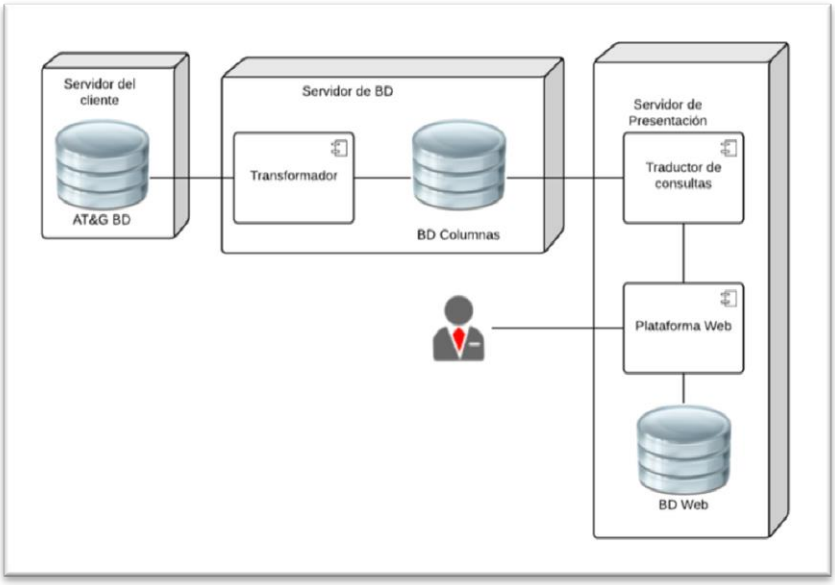

Figura 12-19: diagrama de despliegue

12.5.1.2 Componentes

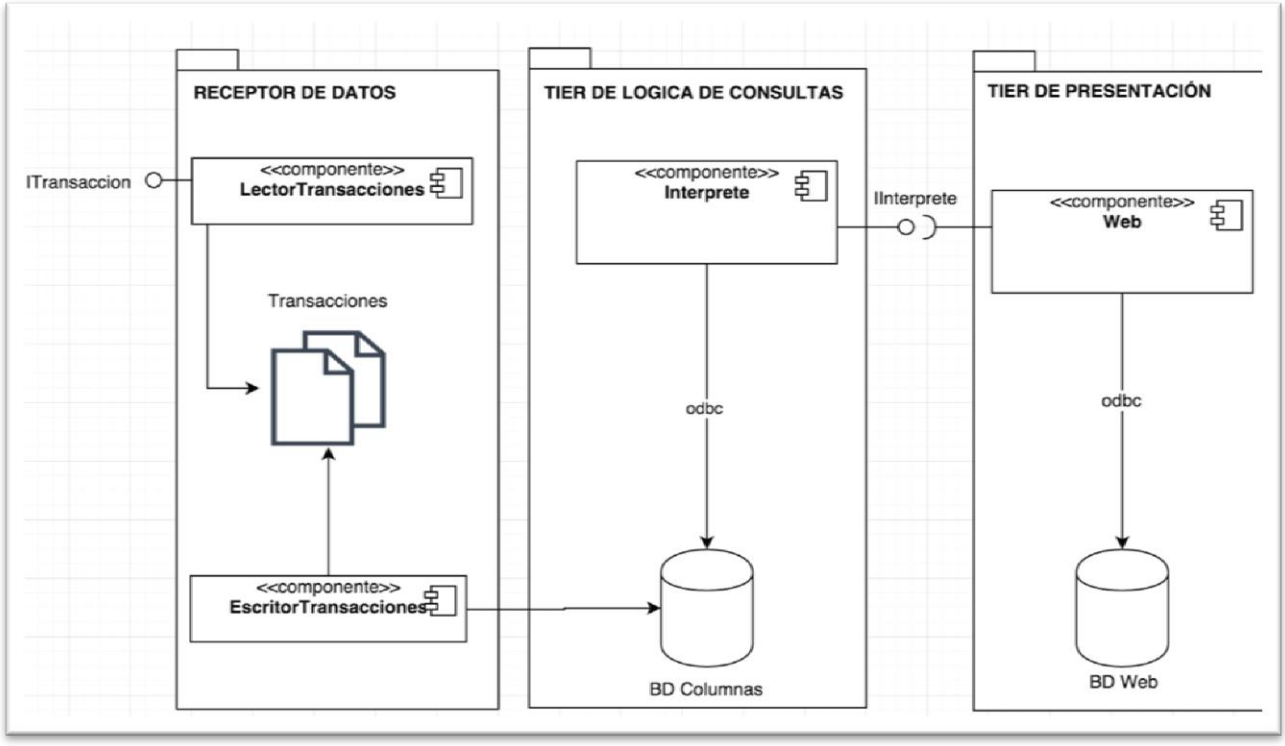

Figura 12-20: diagrama de componentes del traductor de consulta y plataforma web

Descripción de componentes

Componente	Descripción
Lector Transacciones	Encargado de recibir las transacciones enviadas por un sistema de gestión o similar y almacenarlas para un futuro procesamiento. Este componente realiza validaciones mínimas de formato de las transacciones recibidas.
Transacciones	Datos enviados por un sistema de gestión o similar. Estas deben contener los datos que resultan de interés para los usuarios finales.
Escritor Transacciones	Encargado de mantener actualizada la BD en columnas con las transacciones recibidas.
BD Columnas	Base de datos orientada a columnas que almacena los datos

	pre procesados para luego ser consultados.
Interprete	Encargado de implementar las operaciones disponibles en IInterprete.
Web	Página web mediante la cual el usuario final interactuará con el sistema.
BD Web	Base de datos con los datos necesarios para un correcto funcionamiento del componente web.
ITransaccion	Interfaz que se expone al sistema de gestión (o similar) que recibe las transacciones con información relevante para el negocio del cliente.
IInterprete	Interfaz que separa la presentación de datos/información a los usuarios finales, de la definición y lógica del negocio ubicada en el componente Interprete

Tabla 12-59: descripción de componentes del sistema

Receptor de datos es la capa encargada de recibir los datos del ERP y mediante un proceso de transformación insertar los datos la base de datos del sistema. Para esto, se implementó una interfaz que recibe las transacciones que envía AT&G Informática y se guardan localmente en el servidor que las recibe. Luego, mediante un *checkpoint* que permite saber cuáles son las nuevas transacciones, se insertan, actualizan o borran las transacciones, según corresponda, en la base de datos en columnas.

El intérprete será el encargado de traducir las consultas complejas de los usuarios para que luego el manejador de consultas pueda obtener los datos de la base de datos.

Para consultas simples de obtención de datos estándares se utilizará la interfaz IInterprete

12.5.1.3 Módulos

Web

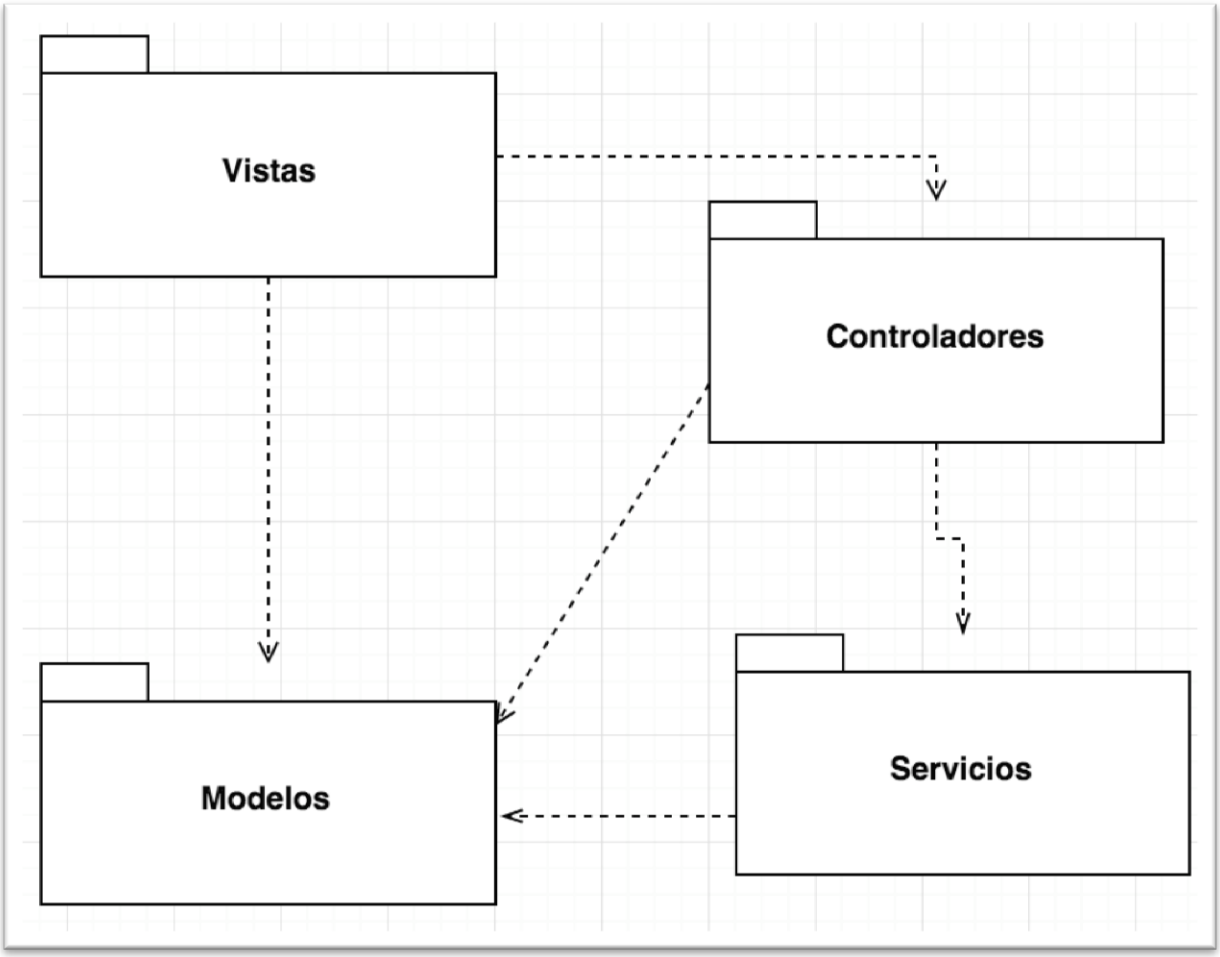

Figura 12-21: diagrama de módulos de plataforma web

Es un modelo simple ya que se buscó que fuese simple de mantener a futuro, y simple de programar en el corto plazo. Principalmente el equipo decidió utilizar una arquitectura MVC ya que era la arquitectura con la que se contaba mayor conocimiento y experiencia.

Interprete

Este componente se encarga de recibir las consultas en lenguaje usuario e interpretarlas a lenguaje SQL.

La creación de la sintaxis SQL fue encapsulada lo más posible de forma que el impacto de cambio en un futuro sea el menor posible

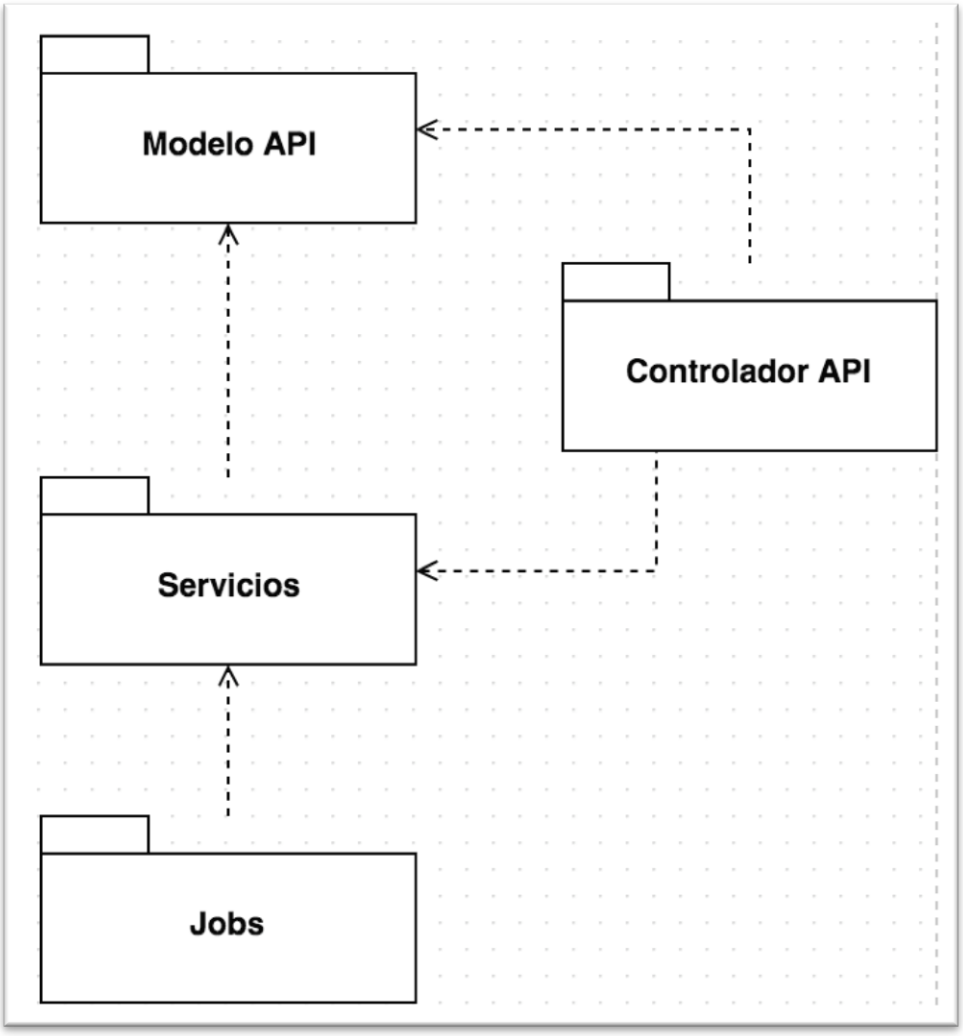

Figura 12-22: diagrama de módulos del traductor de consultas

Controlador API es quien implementa la interfaz expuesta a la Web y mediante el modelo maneja los tipos de objetos definidos en el contrato. Luego utiliza una capa de servicios para realizar la lógica necesaria para procesar los pedidos.

Y por último hay una capa de *jobs* que funciona de forma totalmente asincrónica y se utilizó para realizar eventos o tareas programadas

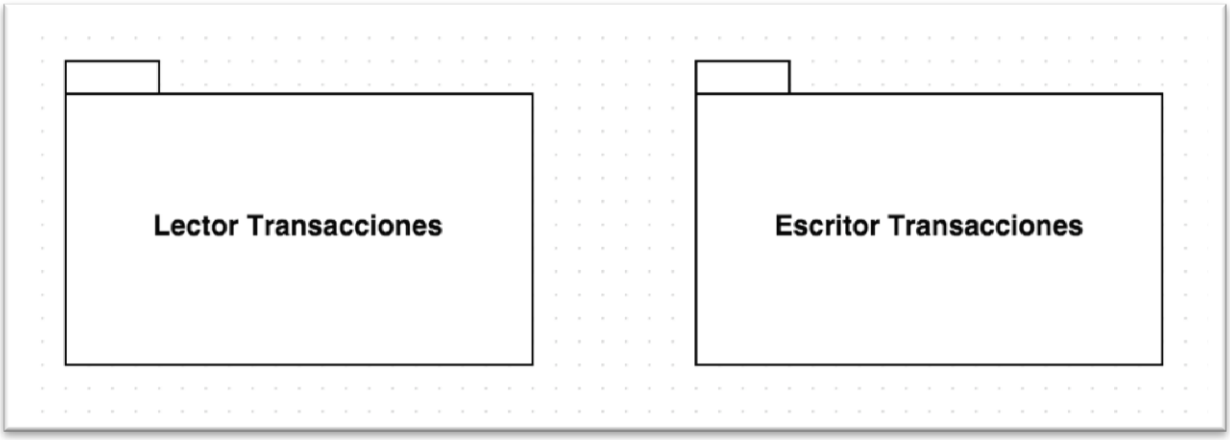

Figura 12-23: diagrama de módulos del Transformador

Lector Transacciones API es la interfaz de comunicación con los servidores del cliente, los cuales nos envían toda transacción nueva, la cual nosotros recibimos y la guardamos en un directorio del servidor propio.

Luego, el Escritor Transacciones es el encargado de, cada determinado tiempo, tomar de dicho directorio todas las transacciones nuevas e insertando en la base de datos propia solo aquellas que pasan el proceso de validación de formato.

Modelo de datos

Gracias a que grails trabaja con Hibernate como ORM por defecto, la similitud entre la estructura de base de datos y el modelo es muy alta. Este ORM utilizado en grails permite entre otras cosas generar la base de datos y mantenerla actualizada desde el código de forma casi automática. Se posee un archivo de configuración *DataSource.groovy*, donde se puede definir el comportamiento de la conexión a la base de datos al momento de iniciar la aplicación, mediante el parámetro *dbCreate*.

Figura 12-24: configuración de los entornos

12.6 Mapeo de datos del dominio y datos para el usuario

Para poder realizar la interacción entre el usuario y el sistema de una forma amigable, intuitiva, sencilla y principalmente en lenguaje usuario, fue necesario utilizar algún mecanismo de traducción de datos.

En concreto se implementó una traducción de los datos que se obtienen desde la interfaz definida para el ERP y los datos que luego se muestran al usuario final.

Mapeos Actuales

Utilizando un mapeo de la base de datos en columnas en estructura de tablas de MySQL se pudo mejorar la calidad de los datos entregados al usuario. Se creó una tabla *rTable* que permite mapear las tablas de la base de datos en columnas con las entidades que el usuario podrá consultar. Con esto se buscó que el usuario pudiese ver entidades que sean familiares para él y no en lenguaje técnico definido por un ingeniero.

Luego para mapear los atributos de las entidades se utilizó otra tabla (*rAtribute*), la cual permite saber que atributo pertenece a que tabla, de esta forma no es necesario que el usuario especifique que desea cruzar datos de varias tablas, o consultar dimensiones. Se encapsula y oculta la información de cómo se genera la consulta al usuario totalmente.

A continuación se deja un ejemplo de cómo se mapean los datos en *rAtribute*

43	Fecha de venta	0	fecha	4	datetime	0
44	CLIENTE - Nombre	0	cliente_nombre	4	varchar	0
45	CLIENTE - Tipo de documento	0	tipodoc	4	varchar	0
46	CLIENTE - Tipo de documento	0	cliente_tipodoc	4	varchar	0
47	CLIENTE - Numero de documento	0	cliente_numdoc	4	int	0
48	CLIENTE - Ramo	0	cliente_ramo	4	varchar	0
49	CLIENTE - Otros	0	cliente_infoadicional	4	varchar	0
50	VENDEDOR - Nombre	0	vendedor_nombre	4	varchar	0
51	VENDEDOR - Tipo de documento	0	vendedor_tipodoc	4	varchar	0
52	VENDEDOR - Numero de documento	0	vendedor_numdoc	4	int	0
53	VENDEDOR - Otros	0	vendedor_infoadicional	4	varchar	0

Figura 12-25: mapeo entre la base de datos MySQL y la base de datos en columnas

Futuros Mapeos

Se entiende que es posible mejorar aún más la forma en cómo se mapean los datos entre la base de datos en columnas y los datos que ve el usuario. Por ejemplo se podría estudiar si es posible evolucionar a una forma de que el sistema aprende de las interacciones del usuario para mejorar la presentación de datos aún más enfocada en el usuario.

12.7 Elementos de configuración de *software*

Nombre	**Tipo**	**Descripción**	**Versión Actual**	**Fecha**
Carta de AT&G	Documento	Carta de acuerdo con AT&G	1.0.3	25/05/15
Presentación de Proyecto	Documento	-	1.0.1	06/04/15
Prueba de Performance	Documento	Pruebas de bases en columnas	1.0.2	21/05/15
Opciones BD	Documento	Opciones de lenguajes para bases de datos	1.0.3	01/05/15

Amazon EC2	Documento	Explicativo del uso de la plataforma Amazon EC2	1.0.0	25/07/15
Estructura	Dato	Estructura de bases de datos	1.1.5	12/12/15
Datos de Prueba	Dato	Datos de prueba de las bases	1.0.7	12/12/15
Interfaz PSIG	Documento	Interfaz definida con PSIG para envío de datos	1.0.8	07/12/15
formato comunicación web traductor.json	Documento	Ejemplo de JSON de comunicación entre módulo web y traductor	1.0.0	03/09/15
Documento de arquitectura	Documento	Detalle de la arquitectura	1.0.4	17/11/15
Documento de Features Set	Documento	Detalle del *feature set*	1.0.5	04/10/15
Gestión de Riesgo	Documento	Detalle de la gestión de riesgos del proyecto	1.0.6	05/12/15
Mock Up's del sitio Web	Documento	*Mock Up's* de la página web del proyecto	1.0.2	15/10/15
Plan de calidad	Documento	Detalle del plan de calidad seguido	1.0.4	03/08/15
Actividades de control de calidad	Documento	-	1.0.4	17/11/15

Plan de *release*	Documento	Detalle del plan de *release* que se realizó	1.0.9	14/12/15
Proceso de SQA	Documento	Detalle del proceso de SQA	1.0.2	29/05/15
Proceso de Desarrollo	Documento	Detalle del proceso de desarrollo	1.0.2	25/05/15
Proceso de gestión	Documento	Detalle del proceso de gestión	1.0.1	29/05/15
Proceso de requerimientos	Documento	Detalle del proceso de requerimientos	1.0.1	29/05/15
Planilla de métricas	Documento	Métricas generales del proyecto	1.0.10	21/11/15
Velocidad del equipo	Documento	Velocidad del equipo en cada *sprint*	1.0.8	21/11/15
ReportesFinal	*Software*	Repositorio del módulo que corresponde al generador de reportes	1.0.9	15/01/16
Traductor	*Software*	Repositorio del módulo que corresponde al traductor de las consultas del generador	1.0.9	27/12/15
ProcesadorTransacciones	*Software*	Repositorio del módulo que corresponde al	1.0.9	02/12/15

		procesador de transacciones		
Transformador	*Software*	Repositorio del módulo que corresponde al transformador	1.0.9	06/12/15

Tabla 12-60: elementos de configuración del software

12.8 Plan de calidad

12.8.1 *Objetivo*

El objetivo de este documento es definir las fases y actividades a realizar durante el proyecto para crear los entregables comprometidos con el cliente.

12.8.2 *Alcance*

Este documento es aplicable al proyecto "Generador de reportes" y a las fases investigación, definición del producto, determinación de features, diseño, desarrollo, pruebas e implantación, y a los procesos de requerimientos, desarrollo y calidad.

12.8.3 Estrategia para el aseguramiento de la calidad

Como marco de gestión se decidió utilizar FDD, el cual al ser parte de la gestión ágil busca ir construyendo la calidad iterativamente e integrarle actividades de aseguramiento de calidad dentro de cada iteración. En concreto, se define un criterio de aceptación en conjunto con todos los interesados el cual debe ser la guía dentro de cada iteración y de dicho criterio deben desglosarse actividades para asegurar la calidad durante cada sprint en las tareas que se realicen. Además, cada integrante del equipo adopta uno o más roles definidos en FDD de forma de oficiar de responsable de que las tareas bajo este rol se cumplan correctamente.

12.8.4 Actividades del control de calidad

Fase	Actividad	Resultado	Rol	Roles	Actividades

			responsable	**participantes**	**de SQA**
Investigación	Estudio de las tecnologías de bases de datos	Documento de investigación	Líder de arquitectura	Desarrolladores	Revisión
	Pruebas de rendimiento	Tabla de comparación de resultados	Líder de desarrollo	Desarrolladores	Verificación
Ingeniería de requerimientos	Extracción de requerimientos	Lista de features	Ingeniero de requerimientos	Cliente, Ingeniero de requerimientos	Revisión
	Especificación de features	Features set	Ingeniero de requerimientos	Ingeniero de requerimientos	Verificación
	Validación de *feature set*	*Feature set* validado	Ingeniero de requerimientos	Cliente, Ingeniero de requerimientos	Validación
Diseño	Diseño de la arquitectura	Documento de arquitectura	Líder de arquitectura	Líder de arquitectura	Verificación
	Validación de la arquitectura	Documento de arquitectura validado	Líder de SQA	Líder de arquitectura, juico experto	Validación
Desarrollo y pruebas	Planificar *feature*	Tareas asignadas a miembros del equipo	Líder de desarrollo	Desarrolladores	Revisión
	Diseñar *feature*	Diseño de *feature*	Líder de desarrollo	Desarrolladores	Verificación
	Construir *feature*	Feature implementada	Desarrolladores	Líder de desarrollo	Revisión, Verificación
	Probar *feature*	Documento de prueba	Líder de *testing*	Líder de desarrollo,	Verificación

		para la *feature*		desarrolladores	
	Integrar *feature*	Feature incorporada al sistema	Líder de desarrollo	Desarrolladores	Verificación, Validación

Tabla 12-61: actividades de control de calidad

12.8.5 Definición de actividades

A continuación se describe en qué momento se realizan las diferentes actividades relacionadas al control de calidad del *software*, con qué frecuencia, responsables y documentación asociada.

12.8.5.1 Revisiones

Producto	Responsable	Resultado	¿Cuándo se realiza?
Documento de investigación (Informe de tecnologías de bases de datos)	Líder de desarrollo	N/A	Luego de terminar de redactar acerca de una tecnología (una revisión por cada tecnología diferente)
Lista de *features*	Ingeniero de requerimientos	N/A	Luego de elaborada una primera versión de la lista de *features*
Tareas necesarias para implementación de una *feature* (Planificación de feature - Desarrollo)	Líder de desarrollo	N/A	Tras haberse determinado todas las tareas para esa *feature*
Feature implementada (Construcción de *feature*)	Líder de desarrollo	N/A	Al finalizar una tarea

Tabla 12-62: tabla de revisiones

12.8.5.2 Verificaciones

Producto	Responsable	Resultado	¿Cuándo se realiza?
Tabla de comparación de resultados (Pruebas de rendimiento)	Project Manager	Informe de verificación de investigación	Al terminar de elaborar la tabla comparativa
Features set	Project Manager	Informe de verificación de features	Luego de documentar los features
Documento de arquitectura	Líder de SQA	Informe de verificación de la arquitectura	Al finalizar la arquitectura
Diseño de *feature*	Líder de desarrollo	Informe de verificación de diseño de *feature*	Una vez planteado el diseño final de la *feature*
Feature implementada (Construcción de *feature*)	Líder de desarrollo	Informe de verificación de implementación	Luego de implementada la *feature*
Documento de *testing* para la *feature*	Líder de SQA	Informe de verificación de *testing*	Al finalizar las pruebas
Feature incorporada al sistema	Líder de SQA	Informe de verificación de integración	Al finalizar la integración

Tabla 12-63: tabla de verificaciones

12.8.5.3 Validaciones

Producto	Responsable	Resultado	¿Cuándo se realiza?
Features set	Cliente, *Project Manager*	*Feature set* validado	Luego de verificar los *features*
Documento de arquitectura	Cliente, *Project Manager*	Informe de validación de la arquitectura	Luego de verificar la arquitectura
Feature incorporada	Cliente, *Project*	Informe de validación de	Al finalizar la

al sistema	*Manager*	*feature*	integración

Tabla 12-64: tabla de validaciones

12.8.6 Testing

Al utilizar FDD como metodología ágil de gestión se cuenta con una etapa de *testing* en cada iteración. En esta etapa el equipo realizará pruebas unitarias para los componentes más críticos del sistema. Luego según resulte conveniente realizará otros tipos de pruebas según la criticidad de las features y componentes de arquitectura que interactúen.

12.8.7 Reporte de problemas y acciones correctivas

Al recibir un problema, ya sea de parte de un usuario o de un miembro del equipo, se deberá evaluar el nivel de impacto del problema en nuestro sistema. Teniendo esto en cuenta se le deberá asignar una prioridad.

Si se detecta durante la iteración en la que se desarrolló, se corrige, mientras que si se detecta en una iteración posterior, se registra como un defecto. La corrección de éste defecto implica un aumento de horas del equipo en la iteración en la que se corrija.

12.9 Evidencias digitales

12.9.1 Minutas de reunión

Las minutas de reunión se dividen en reuniones con AT&G Informática (cliente) y reuniones con el tutor y expertos de dominio.

Dentro de cada uno se encuentran ordenadas de forma cronológica las minutas de reunión.

12.9.2 *Demos* con usuarios finales

En el CD entregado se encuentran disponibles bajo la carpeta Demos las validaciones con usuarios finales.