ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

Actualización del software asociado a un sistema empresarial real en Django

Software updating associated with a real business system in Django

Realizado por
Javier Vázquez Burgos
Tutorizado por
Gerardo Bandera Burgueño
Departamento
Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Noviembre 2014

Fecha defensa:
El Secretario del Tribunal

Resumen: El objetivo del TFG es ejectuar y documentar el proceso de actualización de un sistema software real de carácter empresarial, perteneciente a la empresa dedicada a las transacciones de divisas Foreign Exchange Solutions SL. El sistema está implementado en Python 2.7 usando el framework de desarrollo rápido de aplicaciones web Django que, comenzando por su versión 1.3.1, terminará al final del proyecto en la versión 1.4.10, lo que nos llevará a tener que actualizar todas las librerías relacionadas, además de mejorar la calidad del código e incluso cambiar la estructura del proyecto, prestando además especial atención a la pruebas unitarias y de regresión para comprobar el correcto funcionamiento del sistema a lo largo del desarrollo. Todo esto con el fin de conseguir las nuevas funcionalidades y características que una versión más nueva nos ofrece, además de mejorar la calidad de la aplicación -aumentar la reutilización del código y reducir futuros errores gracias a un código más sencillo y legible-, aumentar el rendimiento, y obtener una buena cobertura de pruebas. Usaremos además la metodología ágil Scrum, el SGBD PostgreSQL, además de otras herramientas como Solr, ElasticSearch, Redis, Celery o Mercurial para el control de versiones.

Palabras claves: Python, Django, Actualización, Pruebas, Documentación, PostgreSQL, Scrum

Abstract: The objective of the TFG is to execute and to document the process of updating of a real entrepreneurial software system, belonging to Foreign Exchange Solutions SL, company engaged in currency transactions. The system is implemented in Python 2.7 using the rapid development framework Django for web applications that, starting in version 1.3.1, until the end of the project in version 1.4.10, which will lead us to have to update all related libraries, and also to improve code quality and even change the structure of the project, paying special attention to the unit and regression tests to verify the correct operation of the system throughout the development. All this in order to get new features and functionality that a new version offers, besides improving the quality of implementation -increase code reuse and reduce future errors thanks to a simple and legible- code, increase the performance and get a good coverage tests. Also will use the agile methodology Scrum, the PostgreSQL DBMS, and other tools like Solr, ElasticSearch, Redis, Celery or Mercurial for version control.

Keywords: Python, Django, Update, Tests, Documentation, PostgreSQL, Scrum

Índice

Acrónimos

BOS	Back Office System
BOS API	BOS Application Programming Interface
EBO	Ebury Online
BDU	BOS Django Update
BSD	Berkeley Software Distribution
DRY	Don't Repeat Yourself
Fx-Solutions	Foreign Exchange Solutions S.L.
FOREX	Foreign Exchange Market
FX	Foreign Exchange Market
SGBD	Sistema Gestor de Bases de Datos
TDD	Test Driven Development
XP	eXtreme Programming
QA	Quality Assurance
LTS	Long Term Support

Capítulo 1. Introducción

1.1. Introducción

Hoy en día la velocidad con la que se dan los avances tecnológicos es cada vez mayor, motivo por el que tanto las empresas como los particulares se ven con la necesidad, incluso obligación en muchos casos, de seguir el ritmo de esta evolución tecnológica. Esto conlleva poner al día el software y hardware de un sistema, lo que puede incluir solucionar fallos o incluir nuevas funcionalidades. Se ha de diferenciar entre dos tipos de actualizaciones: actualización funcional, que consiste en añadir nuevas funcionalidades a las ya existentes, y actualización técnica, consistente en instalar una nueva versión de software.

Es común en la actualidad, aunque cada vez menos, dar prioridad a la funcionalidades, sin prestar demasiada atención a ciertos aspectos como la calidad del código, lo adecuado del software o técnicas usadas, e incluso a la correcta documentación, debido principalmente a la prisas de tener un producto funcionando. Tarde o temprano, esto conlleva un proceso de actualización que puede resultar más costoso que el hecho de usar buenas técnicas durante el desarrollo del producto.

Por lo general, el proceso de actualización comienza con un estudio previo del sistema con el objetivo de descubrir cuales son las carencias del mismo, como pueden ser módulos obsoletos o ilegibles. Posteriormente, y sabiendo ya cuales son los puntos débiles, estudiar los diferentes procesos de mejora, con el fin de elegir aquel más económico, rápido y, sobretodo, factible. Finalmente se lleva a cabo el proceso de migración, una serie de

fases de manera progresiva hasta tener el sistema totalmente actualizado. Es importante comprobar que el funcionamiento de las anteriores funcionalidades del sistema continúan siendo las mismas cuando acabe el proyecto.

Este proyecto se centra en el proceso de actualización de un sistema empresarial real, desarrollado y mantenido por Foreign Exchange Solutions S.L. (Fx-Solutions), una firma enfocada a las transacciones de divisas, especializados en pagos internacionales, en gestión del riesgo asociado a divisas y en proporcionar servicios de gestión de tesorería a empresas en crecimiento.

1.2. Objetivos

El objetivo del presente trabajo de fin de grado es ejecutar y documentar el proceso de actualización de la herramienta web Back Office System (BOS) que está llevando acabo la empresa Fx-Solutions.

Los objetivos del plan pueden dividirse en dos puntos principales:

- Refactorizar el código, con el fin de tener una estructura más ordenada y legible.
- Hacer uso de las nuevas características de las versiones más recientes de Django [1], tanto funcionales como no funcionales.

Con estos cambios se pretenden tres mejoras básicas:

- Menor tiempo de desarrollo, gracias a tener un código más legible y estructurado.
- Aumentar la velocidad en las partes críticas, debido tanto a un uso más eficaz de los recursos, como a las mejoras de las distintas librerías y del propio Django.
- Reducción de errores, debido nuevamente a la refactorización del código y al uso de nuevas técnicas más simples.

La actualización comenzará por la versión de Django 1.3.1 y finalizará con la versión 1.6.1, la última en el momento de redacción de este documento, así como de las distintas librerías implicadas, prestando para ello especial atención a la pruebas de regresión [2], tanto unitarias como de integración y aceptación, para comprobar el correcto funcionamiento del sistema a lo largo del desarrollo.

Es este documento nos centraremos en la actualización hasta la versión 1.4.10.

Capítulo 2. Situación Inicial

En este capítulo nuestro objetivo consiste en describir brevemente el funcionamiento de la empresa y las principales herramientas usadas por Fx-Solutions.

2.1. La Compañía

Fx-Solutions es un proveedor global de software y servicios tecnológicos para la industria de corretaje y pagos internacionales, centrada en el Foreign Exchange Market (FOREX), fundada en 2009 y localizada en Londres y Málaga.

El FOREX puede definirse como la compra y venta simultánea de una moneda por otra. Las monedas son comerciados a través de corredores (brokers) o comerciantes (dealers) y son ejecutadas en pares, por ejemplo, el Euro europeo y el Dólar americano (EUR/USD) o la libra británica y el Yen japonés (GBP/JPY). [3]

A partir de este tipo de operaciones se pueden definir un conjunto de servicios y productos. Fx-Solutions ha desarrollado su propia tecnología para ayudar a las correduría a mejorar su trabajo, reduciendo los errores operacionales y de crédito, soportando nuevos productos y servicios relacionados con el sector, además de mejorar el servicio al cliente.

Al inicio de este trabajo, el equipo de desarrollo estaba compuesto por una plantilla de una decena de programadores, además de un miembro de Quality Assurance (QA)[4] y un Líder de Equipo (Project Leader) a cargo de los distintos proyectos. La forma de

trabajar en la empresa es mediante pequeños grupos de 2 o 3 personas, a veces de un solo integrante, en los que uno de ellos tiene mayor responsabilidad sobre los demás y adquiere el rol de jefe de grupo. Para este proyecto, denominado BOS Django Update (BDU), comenzamos siendo 2 los programadores a cargo.

Cada uno de los miembros del equipo tiene un equipo personal de uso exclusivo con Ubuntu instalado, si bien cada uno elije la versión de este deseada, en nuestro caso nos decantamos por la versión 12.04 por ser la última Long Term Support (LTS), una versión liberada cada dos años que recibe soporte durante cinco años, mientras que el resto son tan sólo de seis meses [5]. Hablaremos de la distintas herramientas y recursos usados en los siguientes capítulos, según vaya siendo necesario.

2.2. Aplicaciones

La principal tecnología desarrollada por Fx-Solutions se denomina BOS, un sistema encargado de llevar toda la lógica de negocio y cuyos principales usuarios son los propios empleados de las empresas de FOREX. A nivel de usuario, el sistema se divide en cuatro secciones principales (imagen 2.1):

- La primera de estas secciones es "Ventas" (Sales), usada principalmente por los comerciales (dealers), encargados de llevar acabo los acuerdos (deals) con los clientes. Esta sección, además de gestionar estos deals, proporciona información en tiempo real sobre las cotizaciones (quotes), contratos (deals) y pagos (payments).
- La segunda sección es "Operaciones" (Operations), usada para gestionar los procesos de fondos (funds) y el flujo de las transacciones, llevar una vista general de todas las operaciones, gestionar las cuentas bancarias y los beneficiarios, y llevar a cabo la reconciliación en los pagos.
- La tercera sección es "Informes" (Reports), que sirve para generar informes de los distintos módulos y tener información en tiempo real sobre los distintos informes y sobre el rendimiento.
- La última de las secciones es "Riesgos" (Risks), encargada de gestionar y analizar toda la información importante que la empresa debe conocer, con el objetivo de minimizar riesgos para la misma.

BOS está implementado en Python 2.7 [6], un lenguaje de programación de código libre e interpretado, orientado a objetos, imperativo y en ocasiones funcional, que te permite trabajar rápidamente e integrar sistemas de forma efectiva, haciendo especial énfasis

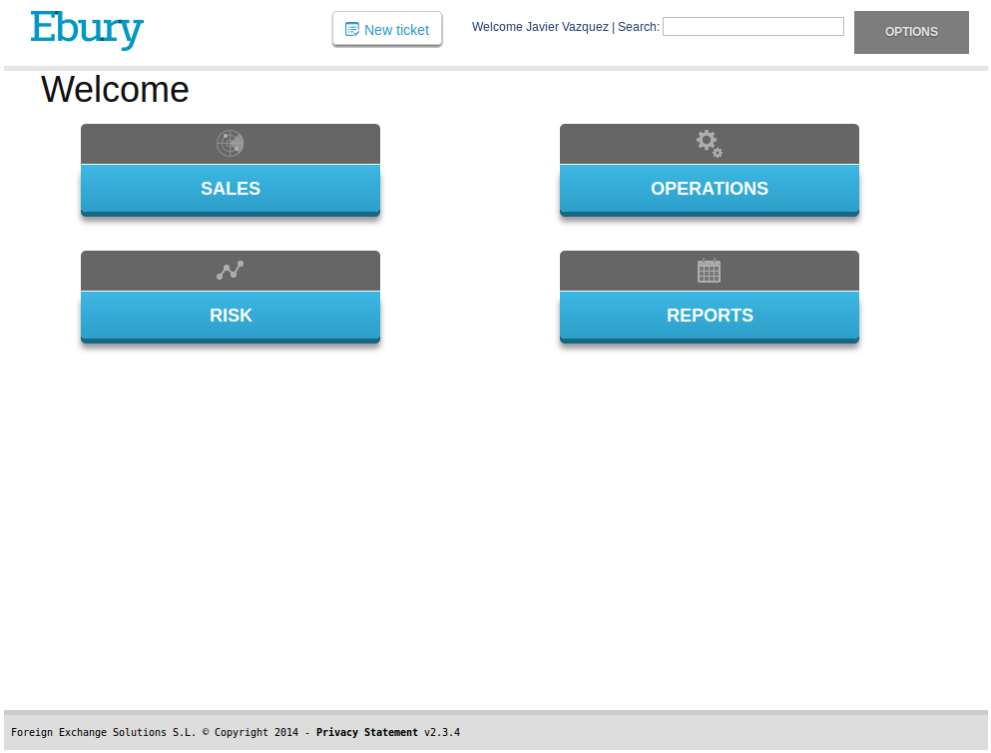


Figura 2.1: Captura de la pantalla principal de BOS

en una sintaxis limpia y legible, lo que lo hace muy sencillo de aprender.

Además usamos el marco de desarrollo (framework) para el desarrollo de aplicaciones web Django 1.3.1, también de código libre, política seguida en todos los desarrollos para Python. Django se trata de un framework de alto nivel que, al igual que Python, fomenta el desarrollo rápido y la sencillez del código, con el objetivo de desarrollar elegantemente sitios webs complejos, gracias al uso de menos código que con otros frameworks más tradicionales. Inicialmente fue desarrollado para gestionar aplicaciones web de páginas orientadas a noticias y más tarde se liberó bajo licencia Berkeley Software Distribution (BSD). Su filosofía se centra en automatizar todo lo posible, mediante el énfasis en su filosofía Don't Repeat Yourself (DRY), que insiste en la ineficacia de la redundancia y los beneficios de la modularización y reutilización del código.

El Sistema Gestor de Bases de Datos (SGBD) usado es PostgreSQL [7], orientado a objetivos, libre, caracterizado por la alta concurrencia y su potencia y, al igual que muchos otros SGBD, soporta joins, claves foráneas, vistas y triggers, entre otras funcionalidades.

Además se usa el motor de búsqueda de código abierto Solr [8] para la indexación de datos y búsqueda de datos (caché) -posteriormente cambiado por ElasticSearch[9]-, basado en Java y con API en XML/HTTP y JSON, el motor de base de datos mediante clave-valor en memoria Redis [10] y el gestor de tareas asíncronas Celery, por destacar alguna de las librerías más importantes.

No vamos a alargarnos demasiado en los detalles de todas las librerías usadas y únicamente en los casos oportunos comentaremos de la forma más breve posible tanto su funcionamiento como su uso.

Volviendo a BOS podemos afirmar que es un sistema realmente basto, y que consta de más de 200.000 líneas de código, 1025 archivos, 851 clases y 8370 funciones, todo esto dividido en 50 módulos, fruto del trabajo durante años de distintos trabajadores, muchos de los cuales nunca han coincidido juntos o no se conocen, y esto ha dado lugar a un código en muchas ocasiones ilegible, en el que cada uno de los programadores ha escrito el código a su manera, en muchas ocasiones de la forma más rápida para salir del paso, sin prestar atención a su futura legibilidad o extensibilidad, con poca modularización, escasa o ninguna documentación, y sin pruebas de ningún tipo para comprobar el correcto funcionamiento. Si bien el ritmo de producción durante años anteriores ha sido alto, el código ha ido creciendo rápida pero desordenadamente por lo anteriormente descrito, lo que hace cada vez más difícil la labor del programador y especialmente para las nuevas incorporaciones que se encuentran con un código difícil de tratar, motivo por el que ha llegado el momento de ser necesaria un actualización del software y una refactorización del código.

Otros sistemas desarrollados por Fx-Solutions son:

- BOS Application Programming Interface (BOS API): Constituye un conjunto de servicios web desarrollados a partir del núcleo de BOS que da servicio a distinta aplicaciones, como por ejemplo Ebury Online.

- FxSuite: Un conjunto de nuevos servicios web especializados para dar soporte a terceras aplicaciones relacionadas con el FOREX, y que sirve de apoyo a BOS.

- Ebury Online (EBO): Un sistema on-line para operaciones de intercambio de divisa que se enfoca a los clientes finales de la empresa Ebury Partners, que pueden acceder para tener conocimiento de sus cuentas y transacciones en BOS, y para solicitar nuevos contratos. Este sistema se soporta sobre la tecnología de BOS API.

2.3. Proceso de desarrollo

El marco usado para la gestión del desarrollo de los distintos proyectos en la empresa es Scrum[11], metodología ágil que sigue un proceso iterativo e incremental. En Scrum tenemos 3 roles claramente diferenciados - y con evidente influencia inglesa-:

- El ScrumMaster, encargado de gestionar los procesos y las distintas tareas.
- El ProductOwner, se corresponde a veces con la figura del StakeHolder, personas interesadas en el desarrollo del software con una serie de requisitos para el producto, los cuales deciden que tareas se han de realizar y con que prioridad - forman una especie de cola ordenada de tareas llamada BackLog-.
- El resto del equipo, en el que están incluidos los programadores, diseñadores, y personal de QA.

En Scrum, un proyecto se suele dividir una serie de etapas (sprints) que pueden considerarse como miniproyectos, y que suelen ser de 2 semanas, si bien el tiempo es variable. Antes de cada sprint deben quedar bien definidas las tareas a realizar, aunque gracias al carácter ágil de esta metodología, no es tan problemático que el ProductOwner cambie requisitos entre sprints: el contacto entre cliente y programador puede ser continuo. Suele exigirse que después de cada sprint el código generado sea un bloque totalmente funcional.

En la empresa seguimos un estándar interno de comenzar llamando al primer sprint con dos palabras que empiecen por A, para luego pasar en el siguiente sprint a la B, etc. Muchas veces se usa un adjetivo y un animal, como Athletic Ant, e ir aumentando de letra hasta terminar el proyecto; en nuestro proyecto concreto solemos acompañar el adjetivo de nombres de serpientes.

En Scrum además se permite la auto-organización de equipos: entre todos los integrantes se estima la duración de las tareas, se deciden que tareas se van a realizar o cuales va a hacer cada uno. Debe existir mucha comunicación en el equipo, y por ello tenemos muchas reuniones diferentes:

- Borrador (Sprint Draft), antes de cada sprint con el fin de planificarlo y decidir las tareas a realizar.
- Reunión diaria (Stand-up meeting), reuniones diarias para el seguimiento del trabajo.

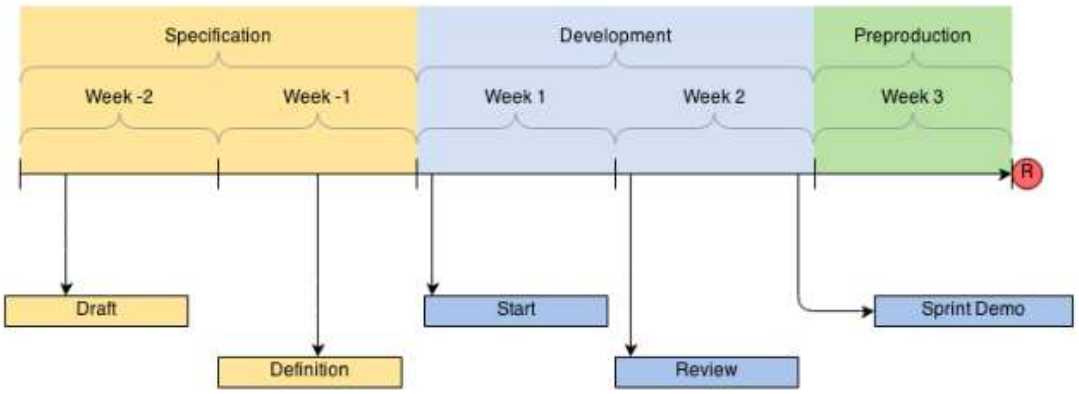


Figura 2.2: Diagrama con el funcionamiento de Scrum

- Reunión de inicio (Sprint Start), al comienzo de cada sprint para decidir que tareas estimar la duración de las tareas, aclarar el contenido y distribuirlo.
- Reunión de revisión (Sprint Review), en mitad y al final del sprint para ver como ha terminado y los problemas que han surgido.
- Reunión de demostración (Sprint Demo), en donde se muestra al Product Owner el producto final de ese sprint.

La organización llevada puede ser observada en las imagen 2.2. Hemos de entender que cuando estamos en la semana 1, estamos la semana -2 del siguiente sprint, es decir, sprint draft y empezar a planearlo; además coincidiría con la semana en la cual se sacará a producción el sprint anterior. Cuando estemos en la semana 2 del actual sprint, estaremos en la semana -1 del próximo sprint, y deberían de quedar cerradas las tareas que se van a realizar en la próxima semana; fijemonos que si todo ha ido bien el anterior sprint debe haber gfinalizado ya. Cuando sea la semana siguiente volveríamos a la situación inicial.

Pueden parecer demasiadas reuniones, pero se ha de intentar agilizar su duración -con medidas como tener la reunión de pie- y sirve para estar todos los afectados por el sprint constantemente informados de la evolución del mismo, lo que nos aporta muchas ventajas, como la posibilidad de corregir rápidamente en caso de error. Además es constante el intercambio de correos y llamadas entre los desarrolladores y los Product Owners en el caso de que no sea posible el encuentro físico entre ellos.

Además usamos ciertas técnicas adoptadas de otras metodologías, como por ejemplo Test Driven Development (TDD)[12]: metodología proveniente de eXtreme Program-

ming (XP)[13], consistente en la realización de las pruebas antes del propio desarrollo del código y la funcionalidad, teniendo en cuenta de antemano aquellas situaciones que nuestra implementación ha de superar. Podemos comentar también el uso regular de la refactorización, del uso de pruebas automáticas y de integración continua.

Estos desarrollos además han de ser coordinados con los diseñadores, el equipo QA y el de sistemas. Por un lado tenemos a los diseñadores, quienes no tienen siempre relación con todos los proyectos, pero en el caso de que sea así, debe asistir también a las reuniones de definición con el fin de comprender las nuevas funcionalidades y poder diseñar los nuevos estilos y gestionar la distribución visual de los elementos con el fin de una usabilidad óptima.

Por otro lado tenemos al equipo de QA, quien también debe asistir a todas las reuniones y cuyo trabajo consiste en definir los diferentes tests unitarios que han de desarrollar posteriormente los desarrolladores -no olvidemos que seguimos la metodología TDD- y los tests de aceptación que ellos mismos harán posteriormente, ya sea con Selenium [14] -herramienta para la automatización de pruebas directamente sobre el navegador, básicamente es un script con una serie de pasos que se van a seguir y una serie de respuestas que se van a recibir- o manualmente; la ventaja del primer método es que el automatizar las pruebas sirve para su posterior utilización como tests de regresión y el poder ejecutarlos repetida y rápidamente una vez implementados. El equipo de QA es el encargado de dar el visto bueno al proyecto y quien dictamina si este sale a producción o es necesario la corrección de errores, el cambio de funcionalidades o la realización de más pruebas antes de ser un producto válido para ser lanzado.

Finalmente está el equipo de sistemas[15], encargado de distintas tareas: por un lado se encarga de la configuración de los diferentes entornos de despliegue -equipos dedicados para cada uno de los diferentes grupos, donde es desplegado tanto BOS como EBO así como los diferentes servicios necesarios-, y por otro lado se encarga de sacar el proyecto una vez finalizado a preproducción (staging) -entorno similar al de producción en donde se prueba durante una semana- y posteriormente sacarlo definitivamente a producción. Las labores tanto de QA como de los SysAdmin son muchas más de las aquí descritas, pero me he limitado únicamente a aquellas que participan activamente con el trabajo de los desarrolladores.

Para el control de versiones usamos Mercurial[16]. Tenemos ramas tanto para producción como para preproducción, además de la correspondiente rama por cada uno de los

proyectos en curso y otra para funciones de soporte. Contamos además con Jenkins[17], aplicación web para la integración continua que nos permite desplegar automáticamente los diferentes entornos y ejecutar los tests cuando se suben cambios al repositorio, además de otras muchas características.

Ya hemos explicado las diferentes metodologías y herramientas usadas durante el proceso de desarrollo, por lo que ya solo queda ir explicando la evolución del proyecto a lo largo de los diferentes sprints hasta tener totalmente funcional BOS con la versión 1.4.1 de Django.

2.4. Por qué Scrum

En muchas ocasiones es cierto el dicho de 'Renovarte o morir' y nunca mejor que aplicado al hecho de tener que actualizar el sistema, y esta vez no va a ser menos hablando de Scrum.

Existen multitud de metodologías para eligir, pero ciertamente Scrum es la más usada y recomendada hoy en día para la mayoría de los desarrollos -tenemos un artículo muy interesante por parte de dos investigadores de Microsoft hablando de las virtudes de esta metodología [18]- y para el proceso de actualización de un sístema más aún.

El hecho de poder ir paulatinamente realizando pruebas (o incluso antes gracias al TDD, algo muy lejos del típico desarrollo en cascada) junto al desarrollo, poder cambiar especificaciones, e incluso sacar versiones del producto no final pero funcional, es una ventaja demasiado abrumadora respecto a las antiguas metodologías.

Llegado el momento es hora de eligir entre Scrum o XP, las dos metodologías ágiles de moda, y es entonces cuando llegamos a la pregunta: ¿Por qué no ambas? Esta fue la opción elegida por nosotros, tener lo mejor de ambas y descartando aquellas opciones que no se adaptan o no son compatibles con el desarrollo. En nuestro caso rechazamos la idea de XP de programar por parejas, ya que aunque es una buena técnica que ahorra bastantes errores, en ese momento la empresa no disponía de tantos recursos. Para ampliar conococimiento sobre el uso coordinado de ambas metodologías, recomiendo 'Scrum y XP desde las trincheras'[19].

2.5. Látex

Este documento describe todo el proceso seguido para llevar a buen puerto el proyecto, y por supuesto para ello nos vemos obligados a hablar de las distintas herramientas que usamos para programar, controlar las versiones, controlar la calidad, etc. Y por esta razón no podemos olvidarnos de Latex, ya que tiene un papel protagonista en todo esto permitiéndonos generar este documento.

Latex es básicamente un sistema de edición de textos, pero enfocado a presentar una alta calidad tipográfica, siendo extensamente utilizado en artículo y libros científicos por parte de físicos, matemáticos o ingenieros [20]. El motivo de su potencia es el estar constituido por comandos TeX (lenguaje con acciones muy elementales). Además es de código abierto, lo que lo hace propenso a recibir librerías de parte de la comunidad -al igual que Django-.

Lo que diferencia a Latex de un editor normal como puede ser LibreOffice, es que mientras en estos ves lo que escribes, en Latex te centras únicamente en el contenido, para luego dar formato al texto mediante etiquetas.

Capítulo 3. Estudio inicial y pruebas de regresión

Comenzamos el proyecto con un primer sprint, llamado Authentic Adder, con dos objetivos claramente diferenciados, por un lado estudiar el estado actual del sistema y de sus librerías y los potenciales cambios a realizar, y por otro lado comenzar a hacer test unitarios que nos servirán en el futuro para las pruebas de regresión. Además nos instalamos el sistema y todas las dependencias y comenzamos a usar Jira y SonarQube [21]. La duración de un análisis inicial puede variar bastante dependiendo del impacto del proyecto y de los miembros a cargo, si bien en nuestro caso fue de tres semanas.

3.1. Estudio Inicial

Respecto al estudio inicial se tuvieron en cuenta dos temas principalmente:

- Librerías: Se elaboraron varios documentos excel para evaluar las diferentes librerías usadas, así como los cambios necesarios y su posible repercusión. El principal de estos documentos podemos verlo en la imagen 3.1 -de manera parcial-, y en el podemos ver una lista con las librerías de Django y de Python usadas, así como la versión inicial y a que versiones era necesario actualizarnos en caso de que fuese posible. Sobre ciertas librerías aparecía poca información en la web por lo que tuvimos que comprobar su correcto funcionamiento en posteriores sprints.

  Usamos un sistema de colores bastante intuitivo: verde si no es necesaria ninguna

actualización, naranja en el caso de que si sea necesaria una actualización, amarillo si no es necesaria actualizar la librería pero si cambiar el código, y rojo si la librería no es compatible. Tenemos un caso más curioso en color fucsia: esta librería es una copia de una funcionalidad de Django 1.4 que no existe en 1.3 y como nosotros la necesitábamos dependíamos de ella, si bien una vez actualizados a la 1.4 podremos dejar de usarla para comenzar a trabajar con la nativa de Django.

- Pruebas (Tests): Es importante planear cuales son las pruebas a realizar y la prioridad, pues por tiempo y recursos es imposible tener analizado el sistema con un 100 % de cobertura, y por ello es necesario decidir que pruebas haremos y cuando. Además es importante cubrir el mayor porcentaje de cobertura posible lo más pronto posible, por lo que es crítico elegir bien las pruebas. Básicamente se creó un excel con distintas páginas, la primera de ellas un cola con las url de las páginas a probar, y el resto de hojas era un resumen por aplicaciones de urls probadas y urls por probar. La idea era ir haciendo poco a poco pruebas sobre las urls de la cola, si bien al final no llegaron a realizarse muchas de ellas.

3.2. Pruebas de regresión

Respecto al tema de las pruebas se realizaron tres tareas principalmente:

- En primer lugar se instaló una aplicación llamada SonarQube para medir la calidad del código y extraer informes sobre el estado inicial de BOS, el cual podemos ver en el anexo 1, si bien en la imagen 3.2, tenemos un fragmento donde se muestra un reparto de las líneas de código en las distintas aplicaciones. Esta información se usará al final del proyecto para ver las mejoras. Obtuvimos información muy variada, desde el número de líneas - 233.465 lineas totales y 185.232 líneas de código -, pasando por el número de archivos -1025- y clases -851-, por código duplicado -21 %-, así como por el porcentaje de comentarios -7.1 %- y por la complejidad, además de un análisis de la distribución del código.

- En segundo lugar se creó una rama Regression a partir de Default (el código de producción), rama en la que se comenzará a hacer los test que el sistema debe pasar después de las modificaciones.

- Finalmente se comenzó a hacer los tests unitarios. En esta etapa se hizo pruebas del login, de la vista de inicio de BOS y de las vistas de inicio de las principales secciones -en Django se llama vista a lo que en Modelo Vista Controlador se llama

Library Name	Library Type	Initial Version	Latest Stable Version(Py.2.7)	Release Date	Django 1.3.1 Max. Version	Django 1.3.7 Max. Version	Django 1.4 Max. Version	Django 1.4.10 Max. Version	Website
Django	Django	1.3.1	1.6.1	12/12/2013	1.3.1	1.3.7	1.4	1.4.10	https://www.dj
django-indexer	Django	0.3.0	0.3.0	25/04/2011	0.3.0	?	?	?	https://pypi.pyt
django-paging	Django	0.2.4	0.2.5	7/05/2013	?	?	?	?	https://pypi.pyt
django-templatetag-sugar	Django	0.1	0.1	24/05/2010	0.1	?	?	?	https://pypi.pyt
django-fixture-magic	Django	0.0.4	0.0.4	8/05/2013	0.0.4	0.0.4	0.0.4	0.0.4	https://pypi.pyt
django-pagination	Django	1.0.7	1.0.7	6/05/2010	1.0.7	?	?	?	https://pypi.pyt
django-endless-pagination	Django	1.1	2.0	28/02/2013	2.0	2.0	2.0	2.0	https://pypi.pyt
south	Django	0.7.3	0.8.4	21/11/2013	0.8.4	0.8.4	0.8.4	0.8.4	http://south.res
django-flash	Django	1.8	1.8	7/03/2011	1.8	1.8	1.8	1.8	https://pypi.pyt
django-extensions	Django	0.7.1	1.2.5	22/10/2013	?	1.2.5	1.2.5	1.2.5	https://pypi.pyt
django-haystack	Django	1.2.6	2.1.0	28/07/2013	2.0.0	2.0.0	2.0.0	2.0.0	https://pypi.pyt
django-celery	Django	3.0.17	3.1.7	11/11/2013	3.1.7?	3.1.7	3.1.7	3.1.7	https://pypi.pyt
django-kombu	Django	0.9.4	0.9.4	1/08/2011	0.9.4	?	?	?	https://pypi.pyt
django-form-utils	Django	0.2.0	1.0.1	19/10/2013	0.2.0	0.2.0	1.0.1	1.0.1	https://pypi.pyt
wadofstuff-django-serializers	Django	1.1.0	1.1.0	1/03/2011	1.1.0	?	?	?	https://pypi.pyt
django-debug-toolbar	Django	0.8.5	0.11	15/11/2013	0.9.4	0.9.4	0.9.4	0.11	https://gemnas
Django Polymorphic	Django	V1.0-RC-1	0.5.3	18/09/2013	0.4	0.4	0.4	0.5.3	https://pypi.pyt
django-taggit	Django	0.10a1	0.11.2	13/12/2013	0.10a1	0.10a1	0.10a1	0.11.2	https://pypi.pyt
django-mptt	Django	0.5.2	0.6.0	6/08/2013	0.5.5	0.5.5	0.5.5	0.6.0	https://pypi.pyt
django-nose	Django	1.2	1.2	23/07/2013	1.2	1.2	1.2	1.2	https://pypi.pyt
django-liveserver	Django	0.1a-2	0.1a-2	3/03/2012	0.1a-2	0.1a-2	Delete	Delete	https://pypi.pyt
pytest-django	Django	2.3.1	2.6	10/11/2013	2.6	2.6	2.6	2.6	https://pypi.pyt
model_mommy	Django	1.1	1.2	7/11/2013	1.2	1.2	1.2	1.2	https://pypi.pyt
django-sentry	Django	1.13.5	1.13.5	4/11/2011	1.13.5	1.13.5	1.13.5	1.13.5	https://pypi.pyt
transdb	Django	0.9	0.9	16/11/2010	0.9	?	?	?	https://pypi.pyt

Figura 3.1: Estudio de las librerías dependientes

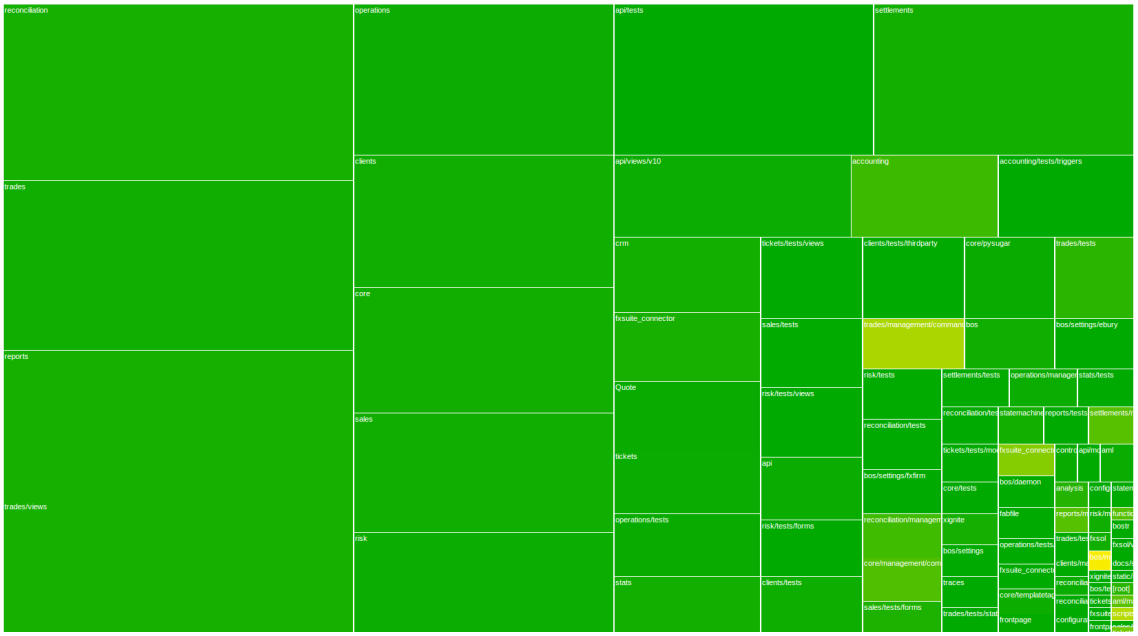


Figura 3.2: Dimensión de BOS por aplicaciones

controlador-. Básicamente eran pruebas sin demasiado nivel de detalle que únicamente comprobaban que al entrar en estas direcciones estando logueados y con los permisos pertinentes, podíamos acceder correctamente y recibir sin problemas los templates - los templates en Django serían equivalentes a la clásicas vistas en MVC, archivos html con código empotrado-.

3.3. Otras tareas

Aparte de las tareas anteriormente descritas:

- Por un lado se creo la rama BDU a partir de la rama Default. En esta rama se comenzarán a realizar todos los cambios propios de proyecto y periódicamente se traerán los tests hechos en Regression para ir comprobando que todo funciona bien. Esto conlleva además la instalación del sistema en local, es decir: descargar los fuentes, crear un entorno virtual para Django 1.3.1 en donde instalarnos todas las dependencias, además de descargar la base de datos y los índices y prepararlos.

  Un entorno virtual sirve para poder tener en una misma máquina diferentes versiones de las dependencias simultáneamente, básicamente lo que hacemos es activar el entorno virtual -al que llamamos bdu_env - y proceder a instalar todas las dependencias -las dependencias son todas aquellas librerías de las que depende la aplicación

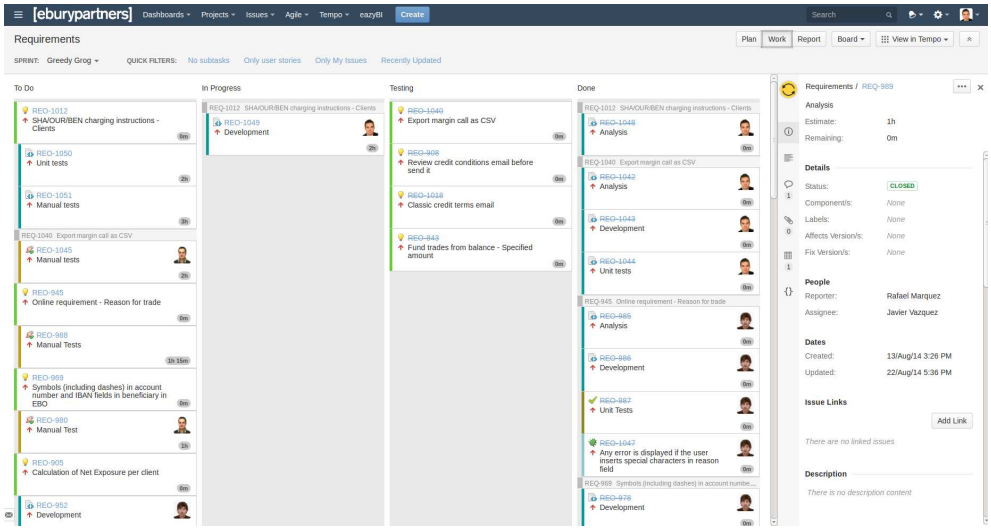


Figura 3.3: Captura de Jira mostrando el tablón de Scrum

para funcionar correctamente, como por ejemplo tener instalado Python 2.7, Django 1.3.1 o Celery -, de manera que mientras tengamos este entorno virtual activo estaremos usando estas librerías, pero en el momento en que salgamos o cambiemos a otro entorno estaremos usando otras distintas, pudiendo así cambiar fácilmente de un proyecto a otro. La herramienta usada para esta tarea es VirtualEnv y realmente es fundamental para casi cualquier desarrollo en Python. Más adelante comenzamos a usar también VirtualEnvWrapper, herramienta que nos permite cambiar rápidamente de un entorno a otro con un simple comando "workon [nombre del entorno]".

- Por otro lado se empezó a limpiar el proyecto, eliminando todo el código comentado o que ya no se usa -tarea difícil, y aún así después de esto seguían quedando demasiadas funciones duplicadas o por las que no se pasaba nunca- y aquellos comentarios que no aportaban nada.

- Durante este sprint y empezando varias semanas atrás comenzamos con el estudio tanto de Python y Django como del resto de librerías y herramientas a utilizar, así como de BOS, pues el conocimiento sobre la mayorías de las áreas necesarias era bastante escaso.

3.4. Al final

Al final de este sprint tuvimos cierto retraso, además de un ligero descontrol, originado por los siguientes motivos:

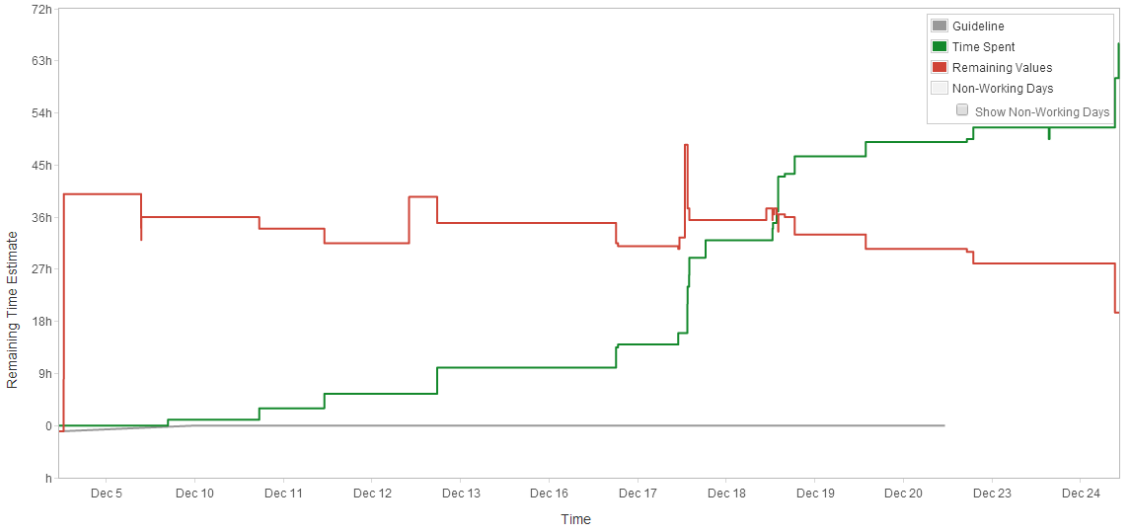


Figura 3.4: Burndown Chart del primer sprint

- Por un lado la poca experiencia de la lógica de negocio, de la lógica de la aplicación, y de las diferentes herramientas a utilizar.

- Por otro lado el ser el comienzo de un proyecto grande dentro una empresa internacional, un contexto en el que existía demasiada experiencia previa, y era una primera fase de toma de contacto.

- Finalmente, el uso e integración de una nueva herramienta para controlar los distintos proyectos: Jira[22]. Jira es una herramienta online integrada con Bitbucket que sirve para llevar un control de los proyectos, sprints, tareas, tiempos dedicados, tiempos restantes, estimaciones, etc.

  En la imagen 3.3 vemos una captura al tablón de Scrum proporcionado por Jira. En el vemos como tenemos cuatro columnas: la primera para aquellas tareas que aún no han comenzado, la segunda para aquellas tareas que están en desarrollo, las tercera para las tareas en pruebas -en el equipo de QA, una vez probada la tarea la cerrará o la pondrá en desarrollo otra vez según su valoración-, y la última columna para aquella tareas que ya han terminado. A la derecha podemos ver una descripción sobre la tarea seleccionada en ese momento -nombre, tiempo restante o quien la tiene asignada-.

En la imagen 3.4 podemos ver Burndown Chart (gráfico de progreso) del sprint. Básicamente vemos dos ejes, en el eje x tenemos los diferentes días del sprint, y en el eje y el tiempo en horas; la línea roja nos marca el tiempo restante que queda hasta acabar

todas las tareas, y la línea verde el tiempo que le hemos dedicado a las diferentes tareas del sprint. Podemos apreciar el descontrol que comentamos antes pues, al ser metidas las tareas una vez iniciado el sprint, el tiempo restante aumenta -algo que no debería ser normal a no ser que nos quedemos sin tareas y añadamos más al sprint-, además de que terminamos sin llegar a acabar todas las estimadas -vemos como la barra roja no llega lleva a tocar el eje x al final de sprint-.

Capítulo 4. Hacia Django 1.3.7

4.1. Introducción

Durante el segundo sprint, llamado Brave Boomslang, que duró cuatro semanas - el tiempo normal estaría en unas dos semanas, pero nos vimos retrasados tanto por la complejidad del código como por la falta de experiencia-, actualizamos el sistema a la versión de Django 1.3.7, así como muchas de las librerías dependientes, además de continuar con la realización de tests de regresión.

4.2. Tareas realizadas

Entremos en detalle en el trabajo realizado:

- Librerías: comenzamos a actualizar las librerías, agrupándolas en tareas según su relación, de manera que quedaron de la siguiente manera: por un lado aquellas relacionadas con los templates de Django -por ejemplo aquellas usadas para paginación, como django-paging o django-endless-pagination-, aquellas relacionados con los test, con pytest-django o model_mommy - está última muy interesante pues nos sirve para crear automáticamente objetos completando aleatoriamente los campos con los datos necesarios (estos objectos son llamados mommies), aunque tuvimos que dedicar un tiempo a pasar gran cantidad de fixtures (modelos con datos completados mediante json) a mommies-, las librerías de Python que necesitaban actualización, así como Polymorphic - librería muy útil para gestionar el polimorfismo en Django-

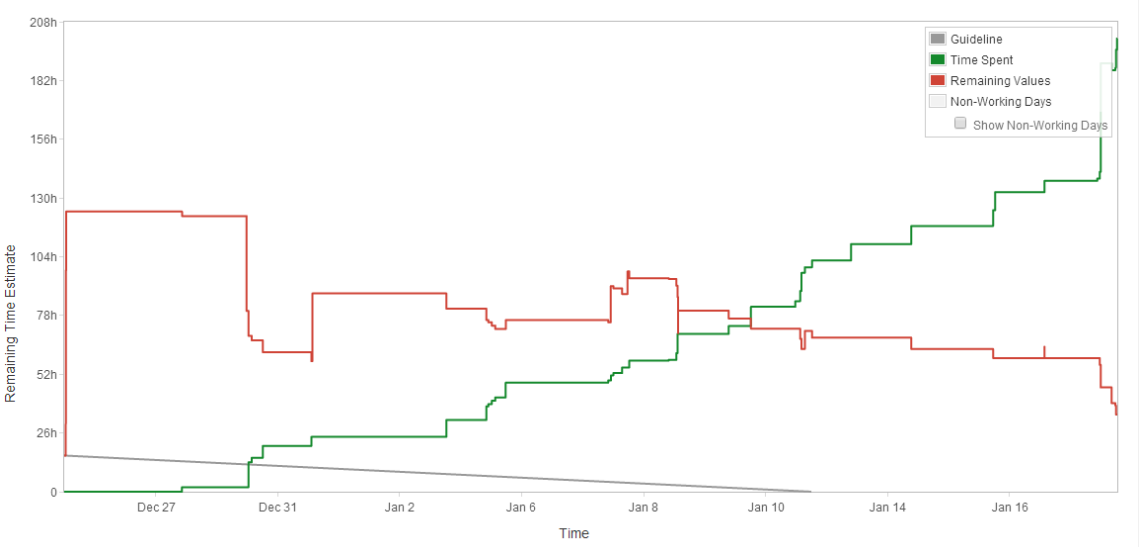


Figura 4.1: Burndown Chart del segundo sprint

de la cual tuvimos que hacer distintos tests para comprobar su correcto funcionamiento, South - librería imprescindible para poder hacer migraciones sobre la base de datos de la manera más segura y fácil posible-, Celery, Haystack - quizás el cambio más destacable, pues pasamos de la versión 1.2.6 a la 2.0.0, lo que supuso gran cantidad de cambios- y Solr - la actualizamos a la última versión compatible con Haystack 2.0.0-.

- jQuery [23]: se realizó además un estudio de los posibles cambios al pasar de jQuery 1.4.4 a 1.9, última versión estable en ese momento, aunque se decidió posponer su actualización para siguientes sprints.

- Pruebas: se continuaron realizando tests, los cuales alcanzaron a crear tests básicos de las aplicaciones risk y reconciliation, y más concretamente del mostrado de ciertos documentos, así como de los contratos de los trades (broker deal), las condiciones de crédito (credit conditions) - las distintas condiciones para los contratos que tienen los clientes -, las margin call - pequeños depósitos adicionales- y de otras áreas más concretas y pequeñas. Además se estudió la inclusión de Model Factory para facilitar la creación de modelos en los tests, si bien finalmente está opción fue descartada.

4.3. Progreso

Al finalizar este sprint teníamos BOS actualizado a Django 1.3.7 y funcionando con total naturalidad. La cobertura de los tests llegó a un 24 % y se redujeron en 2000 líneas más

el número de líneas comentadas. Además de 82 librerías, teníamos 74 -el 90 %- actualizadas a la última versión, si bien es cierto que el número de líneas generales aumentó (+4957) y que la complejidad se redujo muy ligeramente.

Podemos ver en la imagen 4.1 el Burndown Chart de este sprint, en el cual se aprecia que, si bien es cierto que hay mejoría en la organización y en el trabajo respecto al sprint anterior, que aún queda por mejorar.

Capítulo 5. Mejorando BOS

5.1. Introducción

En un tercer sprint, llamado Crazy Cobra, y con una duración de dos semanas -aquí el tiempo si que es completamente variable, dependiendo de la cantidad de recursos que quieras dedicar a la mejora del sistema o continuar el proceso de actualización lo más pronto posible-, añadimos algunos de los cambios disponibles tras actualizar Django a la versión 1.3.7, refactorizamos los tests para disminuir el tiempo de ejecución y de desarrollo de los mismos, creamos nuevas pruebas, comenzamos a usar Jenkins como sistema de integración continua, además de traernos la rama default para no permanecer desactualizados, acción que tendremos que realizar continuamente a lo largo de todo el desarrollo. Finalmente corregimos también algunos errores, pues al termino de este sprint se saco a producción una versión estable de BOS actualizado a la 1.3.7. Después de esto ya quedaba el último y más difícil empujón: actualizarnos a Django 1.4.1.

5.2. Tareas realizadas

Entremos en detalle en el trabajo realizado:

- Unión(merge) con default: un merge consiste en unir dos ramas distintas de código. Puede parecer menos importante de lo que en realidad es, pero en cualquier desarrollo en el momento en el que existen varias ramas y están empiezan a alejarse puede

llegar a complicarse el unirlo en un futuro, por lo que es importante ir trayendo los cambios continuamente para evitar problemas más adelante.

- Refactorización de tests: la inexperiencia hizo que en un principio los tests fuesen más ineficientes de lo necesario, pero sobretodo que estuviesen bastante desordenados. Una de las cosas más importantes que se hizo en este ámbito fue la creación de funciones genéricas para la creación de mommies, fácilmente reciclables y extensibles por los programadores, lo que aumentó el posterior rendimiento del desarrollo. Empezamos a usar marcadores además para ejecutar los tests que nos interesa: podemos marcar los tests más lentos de manera que si es necesario una comprobación rápida podemos ejecutar únicamente lo tests más rápidos.

- Creación de nuevos tests: nos enfocamos sobretodo en la creación de tests de Haystack, pues era un cambio muy delicado y crítico, además la librería se usa en muchas partes y un fallo en una consulta puede ser muy costoso.

- Nuevos cambios: tuvimos que dedicar mucho tiempo a estudiar los nuevos cambios de esta versión tanto en el anterior como en este sprint, para saber distinguir correctamente la importancia e impacto de estos cambios, y saber de que forma proceder. Nos encontramos ante 3 tipos diferentes de cambios:

  - Por un lado cambios de funcionamiento de Django, mejoras ante las que no tenemos que hacer nada. Por ejemplo mejorar la seguridad ante los ataques de Cross-site scripting, o el aumento de formularios inline (formularios dentro de formularios) a 1000.
  - Por otro lado tenemos aquellos cambios o actualizaciones que afectan a funciones o características que no usamos, o nuevas características que decidimos no meter. Por ejemplo tenemos que se evitaron ataques DOS en la subida de imágenes cuando estas son excesivamente grandes, aunque no era un problema que nos preocupase pues no ofrecemos servicio de subida de imágenes; o el cambio de su parseador XML, ya que nosotros usamos una librería externa para esto.
  - Por último tenemos aquellos cambios que forzosamente nos obligan a hacer modificaciones de código, ya sea por cambio de funcionalidad o de uso, o por eliminación de la función y tener que buscar una alternativa. Aunque en este sprint no nos enfrentamos a ninguna tarea de este tipo, en el siguiente sprint y por el mayor cambio de versión si que tuvimos importantes cambios en el código.

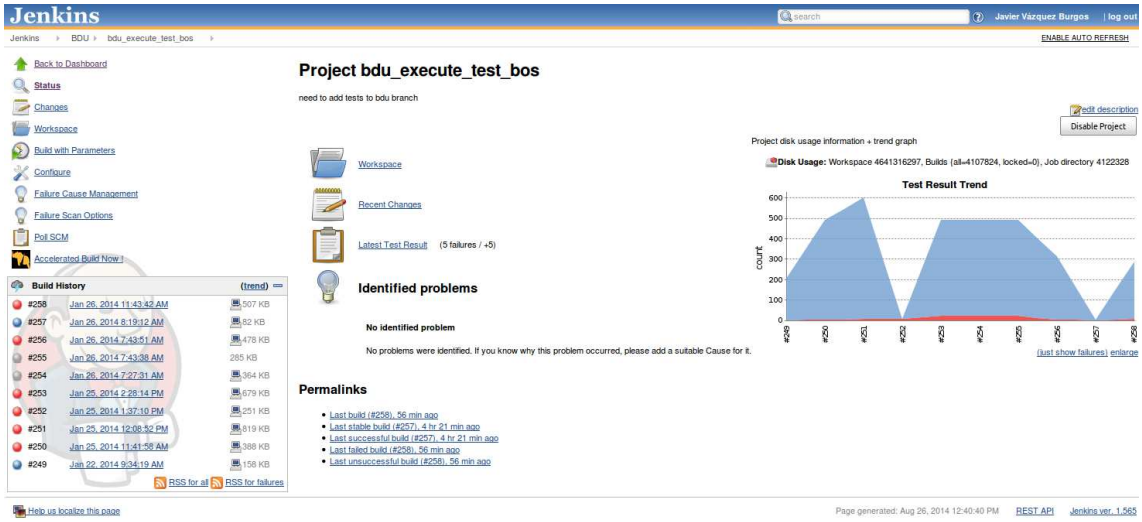


Figura 5.1: Captura Jenkins mostrando la pantalla principal de un proyecto

- Integración continua: comenzamos a establecer las pautas a seguir para el correcto desarrollo y posterior puesta a punto. Para ello comenzamos a usar Jenkins, el cual como ya hemos dicho es un sistema de integración continua, y que nos proporciona diversas funcionalidades, de las que podemos destacar: ejecución automática de los tests cada vez que hay cambios en el repositorio, y así nos aseguramos de no romper nada cada vez que hay un cambio (confianza que va aumentando según aumenta la cobertura y el número de pruebas); y por otro lado permite también automatizar los despliegues en los diferentes servidores de desarrollo y producción. Todo esto lo hace con una interfaz amigable y fácil de usar, de manera que nos permite ver fácilmente si hay tests ejecutándose, cuales fallaron la última vez, llevar un seguimiento de los resultados en el tiempo o reiniciar el servidor con solo un click -imagen 5.1-. Además cada equipo comienza a tener un QA asignado, quien se encarga de gestionar Jenkins, de diseñar los tests de aceptación y tests unitarios que hemos de implementar, y de hacer las pruebas funcionales y manuales correspondientes.

  Volviendo a la imagen, pasemos a describir brevemente lo que vemos. La pantalla nos muestra una visión general sobre los tests ejecutados últimamente sobre el proyecto BDU, del cual podemos ver como en la última ejecución hemos tenido 5 fallos. A la derecha tenemos una gráfica sobre el resultado de los tests a lo largo del tiempo: el color azul determina los tests ejecutados y el color rojo los tests fallados; además a la izquierda también podemos ver el resultado de los últimos tests juntos con algunos detalles. Existen otras opciones, como configurar el proyecto, lanzar los tests o ver los últimos cambios en el repositorio.

- Release (salir a producción): una vez hecho todo lo anterior y comprobado que no hay bugs y que el funcionamiento es el correcto, procedemos a pasar el código a la rama de preproducción. En esta rama está el código durante aproximadamente una semana antes de pasar a producción. Este último paso lo suele hacer el equipo de sistemas, si bien es común que haya uno de los desarrolladores que trabajo en el proyecto por si hay algún problema de última hora.

5.3. Salida a Producción

Hablemos más detalladamente sobre el proceso de sacar a producción un proyecto nuevo. A la hora de publicar un desarrollo nuevo son distintos los enfoques que se le suele dar según la empresa y el tipo de producto, ya que no es lo mismo una herramienta de uso interno a una herramienta pública en la web, ni es lo mismo una sencilla página estática a una aplicación que de servicio a otras o que se encargue de tareas ininterrumpibles.

Por lo general suele ser interesante no cortar el servicio ofrecido en ningún momento, pero no siempre es posible. En el caso de modificar únicamente archivos estáticos es tan sencillo como subir los nuevos al servidor y listo, pero si se requiere cambios en la lógica o incluso en lo modelos, la cosa cambia. Una opción que se suele contemplar en estos casos es la de apuntar la url a otro servidor de manera temporal, y una vez actualizado nuestro servidor principal, volver a redireccionar bien. En nuestro caso, si el cambio es solo de código no hay problema, se actualiza el código y se reinicia el servidor, pero en el caso de que los modelos cambian y haya que modificar la base de datos no podemos simplemente cambiar la url, pues lo cambios que se hagan en los datos en este tiempo no constarán en nuestro servidor principal una vez completada la actualización, además estamos ante un proceso que puede alargarse bastante en el tiempo.

En nuestro caso la opción que solemos tomar es, que en caso de que sea sencillo el cambio -cambiar ficheros estáticos, o código sencillo- se hace un reinicio del servidor a última hora cuando ya nadie está haciendo uso del servicio.

El problema es en el caso de que el proceso sea más complicado, ya sea por cambios en los modelos o por la necesidad de probar el código si es complejo, propenso a errores o las funcionalidad es muy prioritaria; pues esto nos llevaría a tener que invertir tiempo en probar todo, además de tener que hacer una copia de la base de datos y tener tiempo para restaurar el sistema si hay algún problema. En este caso la opción elegida es sacar

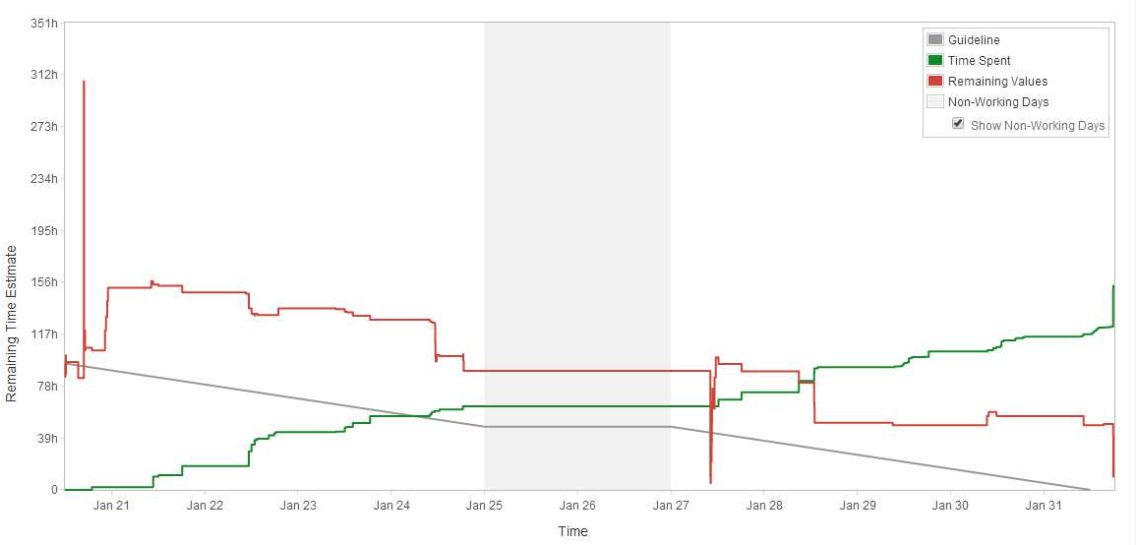


Figura 5.2: Burndown Chart del tercer sprint

el proyecto el Viernes a última hora, ya que al ser una aplicación de uso interno, no es usada durante el fin de semana, y nos da tiempo tanto para probar con seguridad, como para maniobrar correctamente en caso de necesidad.

5.4. Progreso

Al finalizar este sprint teníamos BOS en producción actualizado con todos los cambios realizados hasta ahora en el proyecto. La cobertura de los tests aumentó un 2 % hasta un 26 %.

Podemos ver en la imagen 5.2 el Burndown Chart de este sprint, en el cual ya vemos como el trabajo comienza a funcionar mejorar, y únicamente vemos dos desviaciones puntuales por algún error que fue solucionado inmediatamente.

Capítulo 6. Django 1.4

6.1. Introducción

Hagamos un pequeño balance de nuestro estado actual: nos hemos actualizado y lanzado a producción una versión estable de BOS con Django 1.3.7, actualizando para ello todas las librerías y dependencias; hemos comenzado con la realización de tests unitarios hasta cubrir un 26 % de código; hemos comenzado a usar Jenkins como herramienta de integración continua, además del nuevo apoyo de QA con los tests manuales; y finalmente podemos decir que durante el camino hemos adquirido mucho conocimiento, desde programar con Python y Django, pasando por el uso de metodologías de prueba, hasta llegar al aumento del trabajo en equipo, en parte fomentado por Scrum y su desarrollo ágil.

En el tercer sprint de nuestro proyecto, llamado Dutiful Dugite, de dos semanas de duración -llegados a este punto la división de los sprints no es tan importante, si bien era posible hacer dos sprint de dos semanas, como en nuestro caso, o un sprint de una semana-, nos actualizamos a Django 1.4, que supuso gran cantidad de cambios importantes y críticos. Además de continuar desarrollando tests unitarios, empezamos a diseñar e implementar tests funcionales para cubrir los flujos básicos en BOS.

6.2. Tareas realizadas

Entremos en detalle en el trabajo realizado:

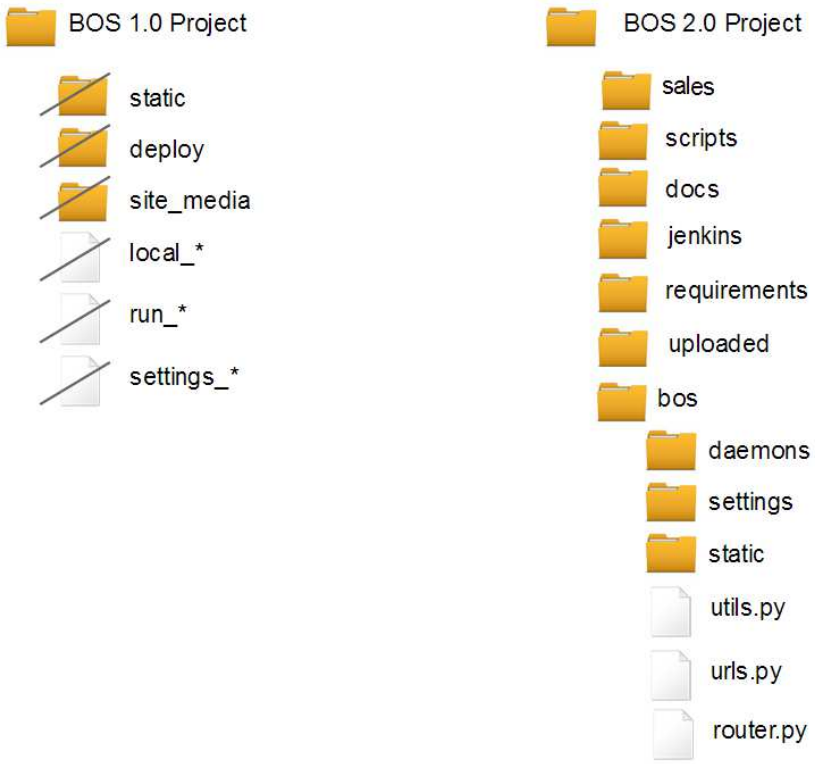


Figura 6.1: Nueva estructura de ficheros después de la actualización

- Actualización a Django 1.4: lo primero fue actualizarnos a Django 1.4, lo que aparte de las mejoras y del cambio en las dependencias, supuso un gran cambio a nivel de estructural. En la figura 6.1 podemos ver la nueva estructura, entre los cambios dados podemos dividirlos en dos: los obligados por Django y los impulsados por nosotros por motivos de orden.

  En el primer grupo tenemos la creación de una carpeta con el nombre del proyecto, y dentro de él tienen que ir el archivo settings.py y el archivo urls.py, en nuestro caso settings es una carpeta porque tenemos diferentes settings dependiendo de en que entorno se despliegue BOS -devel.py, test.py, release.py o prod.py, entre otros, por ejemplo para despegar el servidor de release: python manage.py runserver –settings=bos.settings.release-. Respecto al fichero urls, básicamente hemos de decir que es el fichero que contiene todas las urls de nuestra aplicación o, en muchos casos, la dirección a otros ficheros de urls, pero todas comienzan siempre buscando en este archivo. Otro de las obligaciones impuestas por Django es el tener las carpetas de aplicaciones -digamos que Django se divide en aplicaciones, y dentro de cada una podemos tener su propio fichero urls.py, models.py y views.py, de manera que podemos tener una aplicación para gestionar facturas y otra para gestionar clientes,

si bien todo forma parte al final del mismo proyecto web- al mismo nivel que la carpeta llamada como el proyecto, en nuestro caso bos, como puede ser el caso de la aplicación sales.

Los cambios incentivados por nosotros son el tener una carpeta docs dentro de la cual tenemos toda la documentación -más adelante hablaremos de esto en detalle-, otra carpeta con todos los scripts -para crear la base de datos de tests, para hacer una reconstrucción (rebuild) de los índices-, una carpeta requirements - que tiene los archivos con las dependencias para cada uno de los sistemas, por ejemplo para instalar las librerías de desarrollo: pip install -r requirements/dev.txt-, o el inclusión de los ficheros estáticos (css, js, imágenes) dentro de la carpeta BOS, de los daemons -los daemon son procesos que se ejecutan periódicamente o continuamente encargados de ejecutar ciertas funciones- o del fichero utils.py -que contiene una serie de funciones creadas por nosotros que nos son de mucha utilidad para usarlas en las diferentes aplicaciones-. En la raíz del proyecto encontraremos también el archivo manage.py, uno de los más importantes de Django, pues es el encargado de ejecutar gran cantidad de los comandos -python manage.py command-, cuyo código hay que cambiar, pues en Django 1.5 la versión antigua es marcada como deprecated (en desuso) y en Django 1.6 ya no puede usarse.

Estos cambio estructural llevo además un gran cambio de código, pues muchas importaciones hubo que refactorizarlas -por ejemplo from utils import * pasó a ser from bos.utils import *, o import settings.py pasó a ser from django.conf import settings, que toma automáticamente el settings con el cual se ha lanzado el proyecto-.

Por último comentar que las variables con datos sensibles como contraseñas para la conexión la base de datos o con servicios externos fueron sacadas del código por motivos de seguridad, de modo que era necesario tenerlas activas en el entorno virtual de la manera habitual de Linux:

export BOS_DB_DEFAULT_USER=*****

export BOS_DB_DEFAULT_PASSWORD=*****

- Actualización de librerías: algunas librerías tuvieron que ser actualizadas:

  django-form-utils 0.2.0 to 1.01

  django-polimorphic 0.4 to 0.53

django-nose 1.2 with changes in configuration

remove django-liveserver

psycopg 2.5 to 2.5.1

La peor parte vino con transdb, ya que por la actualización de Django se hizo incompatible, de modo que tuvimos que cambiar el código fuente de la librería.

- Otros cambios: como siempre, vienen una serie de cambios y mejoras con Django los cuales no implican nada por nuestra parte, ya sea por mejorar internas del framework o por cambios en funcionalidades que no usamos, como por ejemplo:

  - El cambio del sistema de hashing, ya que hasta ahora veníamos usando SHA1, pero a raíz de la actualización se comenzará a usar PBKDF2. Los nuevos usuarios se crearán de esta manera y para los antiguos usuarios se comenzará a usar el nuevo sistema cuando acceden al sistema por primera vez después del cambio.
  - Se crea una nueva clase django.test.LiveServerTestCase que permite mejorar la interacción entre los tests de la parte backend y la parte frontend, si bien no es algo que nosotros usemos actualmente.
  - Algunas funciones, como django.contrib.auth.models.check_password, han sido movidas de sitio o cambiadas de nombre, pero son funciones que nosotros no venimos usando.
  - Se añadió la posibilidad de reiniciar la contraseña desde la interfaz de administrador únicamente añadiendo la vista admin_password_reset con la url deseada en nuestro urls.py
  - Se comienza a exigir una SECRET_KEY, variable única y secreta para mejorar el sistema de seguridad de Django, si bien nosotros teníamos ya una de antes.
  - Otros muchos cambios, como con MySQLdb o flatpages, que no son usados en BOS, o que la ejecución del runserver es ejecutada multihebra por defecto.

  Además tenemos otra serie de cambios los cuales se requirieron esfuerzo por nuestra parte, como por ejemplo que Django comenzó a dejar de usar unos archivos de imágenes de la administración, archivos que nosotros si que usábamos en ciertas partes de BOS, por lo que tuvimos que sustituirlos.

- Seguimos con las pruebas: por un lado se diseñaron gran cantidad de pruebas funcionales y unitarias sobre partes importantes de BOS, como son los balances, la

Figura 6.2: Burndown Chart del cuarto sprint

máquina de estados -ciertos objetos en BOS, como los contratos y los pagos están relacionados a una máquina de estado, de modo que van transitando por ellos desde que son creados hasta que ya son ejecutados, pudiendo ser bloqueados o cancelados, entre otros tantos estados-, el acceso, ciertas llamadas ajax, y algunos formularios. Además se implementaron tests funcionales sobre el algunos flujos básicos en BOS, como la creación de un contrato. Finalmente se implementaron los tests diseñados sobre los balances, parte realmente importante en BOS, pues no se puede tolerar que haya un error en la cantidad que un cliente tiene, debe, o ha pagado.

- Bugs: durante el sprint surgieron gran cantidad de errores que hubo que ir solucionando: tests que dejaban de funcionar, templates que dejaban de cargar campos de texto, conexiones con servicios externos que dejaban de ir correctamente, imposibilidad de acceder a la página de administrador, y un largo etc. Parte de la culpa la tiene el merge habitual con default, el cual suele generar conflictos y bugs.

6.3. Progreso

Realmente este fue un punto importante, ya que conseguimos llevar a BOS a la versión 1.4.1 y poderlo lanzar de manera estable en local y en los servidores de desarrollo. La cobertura de tests se mantuvo en un 26 %.

Podemos ver en la imagen 6.2 el Burndown Chart de este sprint, y como tuvimos un claro error de estimación, además de vernos abrumados por la gran cantidad de errores

que nos encontramos y la magnitud de ciertos cambios como el de la estructura de los archivos. No obstante, el error de estimación no fue tan grande como el mostrado en la gráfica, sino que tuvimos problemas con Jira con el tiempo estimado de las tareas y subtareas que no bajaban del sprint según se terminaban sino según se cerraba la Story User, de modo que la gráfica no es un reflejo de la realidad del sprint.

El siguiente paso sería actualizarlo a la versión 1.4.10 de Django -siguiente capítulo y sprint-, para finalmente dejarlo totalmente estable y con todos los detalles pulidos para la puesta en producción -último capítulo y sprint del proyecto-. Aun queda cambios importantes, como el cambio de Solr por ElasticSearch, lo cual supuso más de un dolor de cabeza.

Capítulo 7. Django Desencadenado

7.1. Introducción

Durante el quinto sprint, Euphoric Elapidae, de dos semanas de duración, llegamos a actualizar a Django 1.4.10, última versión de Django 1.4 y versión en la que quedará finalmente el proyecto -y de ahí el característico nombre del sprint-, añadimos un sistema de documentación al proyecto, y además continuamos con algunas mejoras y nuevos tests como hasta ahora.

7.2. Tareas realizadas

Entremos en detalle en el trabajo realizado:

- Actualización a Django 1.4.10: lo primero nuevamente fue actualizar BOS, esta vez a Django 1.4.10, cambio de menor embergadura al anterior pero no por ello menos importante, pues se trata de la última actualización y hay que realizarla con el cuidado correspondiente, si bien es cierto que no conllevó grandes cambios directos, aparte de mejoras internas de Django.

- Actualización de librerías: Las librerías actualizadas fueron:

  django-debug-toolbar 0.9.4 to 0.11 - django-debug-toolbar es estable a partir de la versión 1.0, pero esta necesita de jQuery ¿2.0 y como en BOS tenemos 1.4.4 no podemos actualizar hasta el final. En el siguiente sprint se actualizará jQuery.

Django Polymorphic 0.4 to 0.5.3

django-taggit 0.10a1 to 0.11.2

django-mptt 0.5.5 to 0.6.0

- Mejoras: nuevamente se realizaron una serie de mejoras, como apoyo a las zonas horarias, ya que antes era trabajo del programador gestionar las diferentes horas mostradas a los usuarios según su localización geográfica, mientras que ahora Django te proporciona un sistema de horarios gestionado por él. Aún así este cambio no nos afectó, pues desde Django 1.3 está función ya está incluida en aquellos sistemas con PostgreSQL como SGDB, el cual guarda en la base de datos las fechas necesarias y luego son traducidas a la hora de mostrarse de manera correcta a cada usuario.

  Otras cambios fueron añadidos, como mayor compatibilidad con HTML5 en la vista de admin, mejoras de funcionamiento y seguridad en la vista de admin, la posibilidad de usar un reverse perezoso, mejor apoyo a IPv6, una nueva clase SimpleTestCase más ligera que TestCase aunque con menos funcionalidades, además de varias mejoras de seguridad, por mencionar algunos de los cambios.

- Documentación: se introdujo un sistema de documentación con Django mediante Sphinx, el cual esta basado en docutils. Realmente el uso de Sphinx para generar la documentación es bastante sencillo, y permite exportar el texto plano a html o pdf, por nombrar un par de formatos.

  Sphinx tiene su propio lenguaje de marcado el cual es muy fácil de aprender, y una vez desplegado el código podemos acceder a el vía web con un estilo muy similar a readthedocs. En la captura 7.1 podemos ver un ejemplo.

- Tests: en este sprint se trabajaron bastante los tests:

  - Por un lado se hicieron tests sobre los demonios -tareas que se ejecutan en segundo plano, basicamente es un comando de Django el cual se programa para que se ejecute cada cierto tiempo o que está corriendo permanentemente en segundo plano-, si bien inicialmente solamente se testeo la clase padre de la cual heredan los demás.
  - Por otro lado se testeo la máquina de estado e, igualmente que en el caso anterior, se testeo únicamente la clase padre. En BOS tenemos una serie de maquina de estados para controlar como evolucionan los distintos contratos y pagos, así como las condiciones para ello, y era bastante prioritario tener testeado y controlado que esto se realizase de manera correcta.

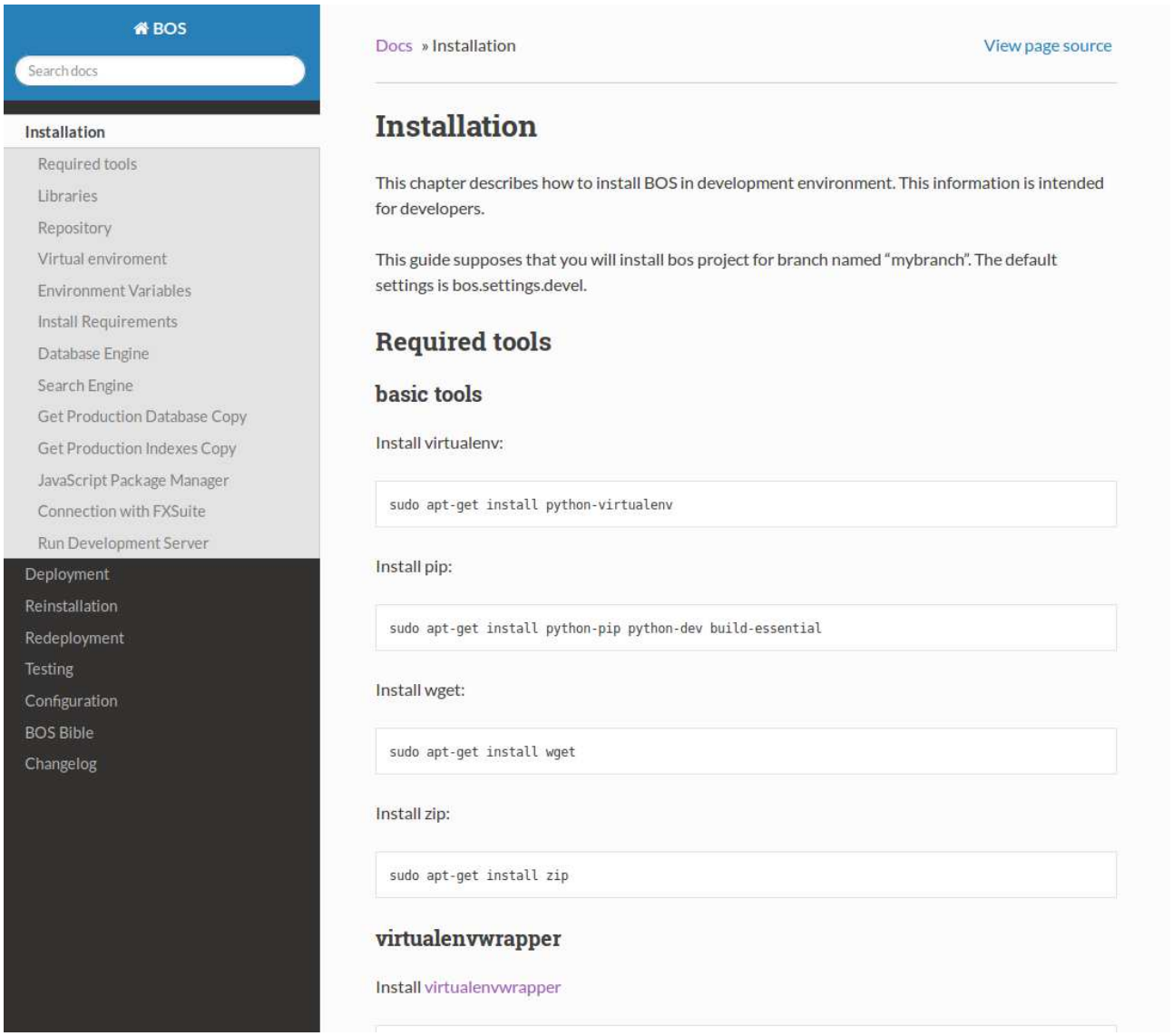


Figura 7.1: Captura de la documentación del proyecto

- Además, se creo un script para comprobar, tanto en local como en los diferentes entornos, que la conexión con los distintos servicios -xignite, el cual nos proporciona los ratios de cambio; o sugar, nuestro crm; por nombrar algunos- se realiza correctamente.
- Continuamos también intentado aumentar la cobertura de tests y para ello realizamos pruebas de la parte de reconciliación, modulo se encarga de unir aquellas entras bancarias reales que llegan al sistema con las entras en BOS, parte importantísima como es de imaginar.
- Por último, cambiamos el SGBD para los tests, los cuales se venían ejecutando con Sqlite -bastante ligero- por PostgreSQL -más lento pero mucho más potente-. Básicamente tuvimos que cambiar la configuración en el settings, además de modificar nuestro script que nos creaba la base de datos de tests. Se aprovecho para mejorar algunos tests que comenzaron a fallar y que sqlite no detectaba.

- La última de las tareas y la que llevó más tiempo fue la corrección de errores no tanto por los cambios realizados en este sprint sino por la actualización del sprint anterior, el cual dejó muchos bugs pendientes. Hubo que corregir prácticamente todos los tests, los cuales pasaron de heredar de TestCase a heredar de BosBaseTest -esta nueva clase hereda de la anterior, con la diferencia de que antes de ejecutar cada test reinicia los índices al estado inicial-. Referente a lo anterior, algunos filtros que usaban los índices comenzaron a fallar también, y nos dimos cuenta que algunos iban más lento de lo deseado, motivo por el que comenzamos a pensar en cambiar de motor de índices, si bien se hicieron algunos fixes temporales. No podemos olvidarnos además del merge habitual y de los problemas que trae.

7.3. Progreso

Llegamos a los últimos momentos del partido y aunque pueda parecer que ya está todo listo, en el siguiente y último sprint se cambio Solr por ElasticSearch, cambio bastante complejo, además de mejorar la documentación, optimizar el rendimiento de pantallas críticas en BOS, actualizar jQuery, así como terminar de hacer nuevos tests y mejorar algunos antiguos, y corregir bugs. La cobertura de tests subió a un 27 %.

Podemos ver en la imagen 7.2 el Burndown Chart de este sprint, nuevamente atacados por la gran cantidad de errores -principalmente los índices-, motivo que hizo que si bien

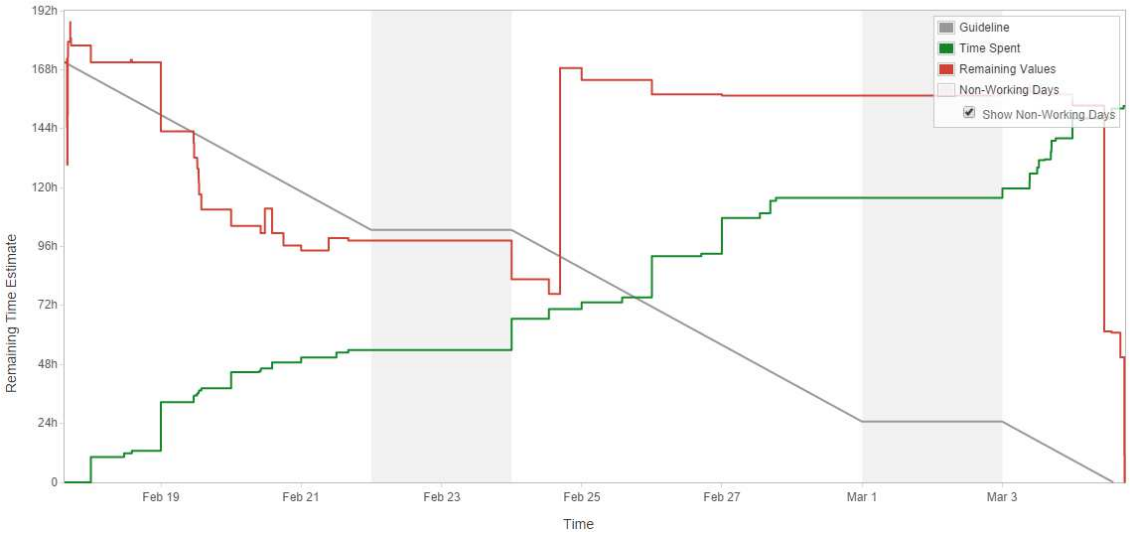


Figura 7.2: Burndown Chart del quinto sprint

se tenía pensado acabar el proyecto en un sprint de tres semanas, se pasó a dejar el sprint en dos semanas y terminar con un último sprint de otras dos semanas que dejase todos los detalles finos y que nos permitiese cambiar a ElasticSearch. Cabe decir que el motivo de la repentina subida a mitad del sprint fue la incorporación de test funcionales por el equipo de QA que previamente no habían sido estimados, por lo que supuso un aumento del trabajo restante que no se había pensado inicialmente.

Capítulo 8. Finalizando el Proyecto

8.1. Introducción

Durante el último sprint, Forceful Flying-Snake, de dos semanas de duración (nuevamente podría haber sido ampliado el plazo o incluso posteriores sprints para añadir más mejoras, si bien se decidió finalizar el proyecto aquí), conseguimos dejar BOS totalmente estable, sin errores, y con mejoras de rendimiento funcionando perfectamente en todos los servidores. Actualizamos jQuery, mejoramos la documentación, y cambiamos Solr por ElasticSearch.

8.2. Tareas realizadas

Entremos en detalle en el trabajo realizado:

- Actualización de jQuery: un cambio que se comió también bastantes horas fue la actualización de jQuery, la cual se hizo en dos pasos. Primero actualizamos de 1.4.4 a 1.11.0, para posteriormente actualizarnos a las 2.1. Lo que se hizo fue testear manualmente las distintas páginas para corregir aquellas que dejasen de funcionar o que cambiasen de comportamiento.

- Tests: se continuó realizando tests funcionales de los flujos principales, además de realizar algunos tests nuevos para comprobar los correctos cambios de jQuery. En este sprint no se realizaron tests unitarios.

- Documentación: Se mejoró la documentación añadiendo un manual de instalación para todos aquellos que quisiesen actualizarse BOS al estado posterior a este proyecto o para aquellos que quisiesen instalarlo de 0.

- Integración de ElasticSearch: lo primero fue cambiar los distintos settings y las dependencias, y luego se comenzó a corregir los distintos errores que fueron surgiendo. Además hubo que cambiar los diferentes scripts existentes y documentar las novedades.

- Securizar el setting: lo que hicimos fue sacar todas las variables sensibles del setting, pues tanto como contraseñas como identificadores era puestos directamente sobre el código. Lo que hicimos fue que se extrajeran de las variables de entorno, por lo que cada uno en su entorno era responsable de tener las variables correctamente actualizadas.

- Optimizar pantallas críticas: se hizo un gran esfuerzo en aumentar la velocidad de carga de aquellas páginas demasiado pesadas o importantes. Uno de las características que se intentó usar fue la paginación en aquellas partes más antiguas que no fueron inicialmente hechas así. Se intentó además reducir el tamaño de la base de datos, pero debido a lo entremezclado de los datos, y a la necesidad de sacar reportes sobre diferentes datos muy antiguos o incluso desde el comienzo de la aplicación, hicieron inviable esta opción.

- Errores: por supuesto este sprint tuvimos que corregir muchos errores, si bien es cierto que el merge fue más ligero que de costumbre. Tuvimos que pelear con Jenkins, el cual fallaba al intentar matar a celery si no estaba andando (se solucionó con un '— true' en el script); se mockearon - un mock es un sustituto de una función que devuelve un valor predefinido- ciertas llamadas a servicios externos, pues algunos tests comenzaron a fallar por motivos ajenos a nosotros, ya sea por tiempo excedido (timeout) o por incapacidad de conectar en ciertos momentos; por comentar algunos de los errores más interesantes.

- Mejoras de rendimiento: se instaló en un sistema BOS antes de la actualización y en otro después de todo el proyecto para comparar los resultado de los diferentes tests de escalabilidad y de rendimiento. En el anexo 2 encontramos el documento que se extrajo del análisis, en el podemos ver algunas páginas en las que el aumento del rendimiento ha sido más que notable, aunque también hay algunas en las que se ha reducido algo el rendimiento, aunque puede ser por el aumento de información

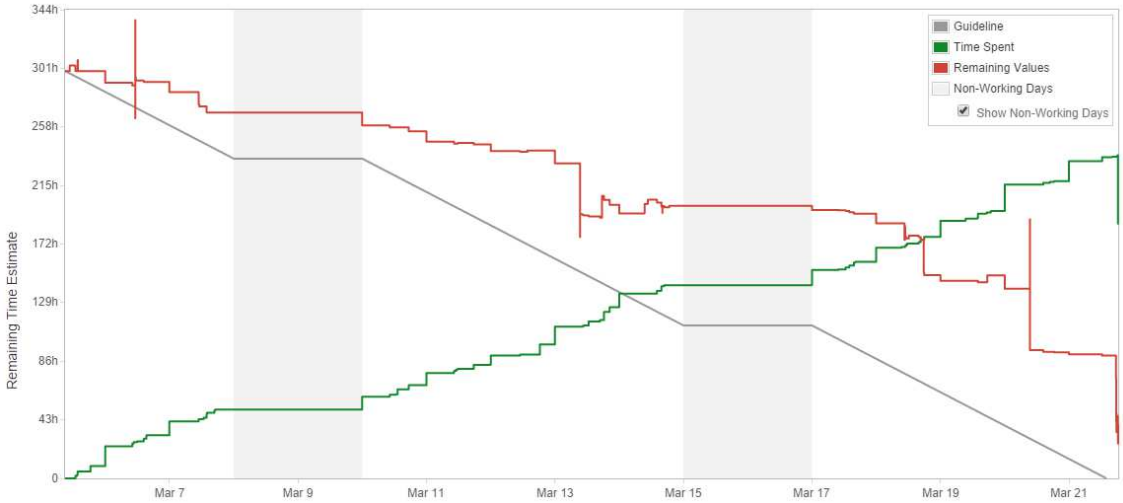


Figura 8.1: Burndown Chart del sexto sprint

de la base de datos, o por las nuevas funcionalidades que han ido añadiendo otros equipos paralelamente a nuestro desarrollo.

- Release: después de todo este desarrollo se tuvo al proyecto una semana más en preproducción para terminar de probar la aplicación y corregir los últimos detalles, hasta que finalmente se puso en producción sin grandes complicaciones.

8.3. Progreso

Finalmente hemos finalizado el proyecto, dejando BOS en la versión de Django 1.4.10 de manera estable. En el siguiente capítulo analizaremos la evolución del proyecto y el estado final.

Podemos ver en la imagen 8.1 el Burndown Chart y como nos hemos acercado más a la estimación inicial. Solo nos hemos dejado en el tintero el realizar más pruebas funcionales del flujo de la aplicación, pero aún así estamos bastante contentos con el resultado final.

Capítulo 9. Conclusión

9.1. Resumen

Nos hemos encontrado con 6 sprint a lo largo de los cuales hemos ido actualizando Django y las diferentes librerías, haciendo los oportunos cambios de código e incluso de tecnología a veces, para terminar dejando el sistema en un estado estable.

En el primer de estos sprint hicimos un estudio fundamental de estado inicial del sistema y de las diferentes pruebas a realizar, además de la posibles librerías a actualizar, y creamos los distintos entornos necesarios. Comenzamos con la implementación de pruebas unitarias y limpiando un poco el código, además de empezar a usar Scrum como metodología Ágil con la herramienta web Jira.

En el segundo sprint, aparte de seguir desarrollando más pruebas, actualizamos Django y las librerías asociadas, además de estudiar la actualización de jQuery.

En el tercer sprint seguimos desarrollando nuevas pruebas, e incluso se mejoraron los ya existentes. Importante en esta etapa fue la inclusión de la Integración Continua y el uso de Jenkins. Al final de este sprint se saco a producción una primera versión estable con los primeros cambios.

En el cuarto sprint seguimos naturalmente con las pruebas unitarias, pero también comenzamos a hacer pruebas funcionales. En esta fase se dio el paso más grande de versión,

hasta Django 1.4.1, lo cual supuso una enorme cantidad de cambios y bugs.

En el quinto sprint hicimos la actualización final de Django y de algunas librerías, y añadimos un sistema de documentación con Sphinx y continuamos analizando los cambios y corrigiendo errores. Hubo gran movimiento en el tema de tests unitarios, sobretodo teniendo en cuenta el cambio de Sqlite a PostgreSQL.

En el sexto y último sprint, dejamos BOS totalmente estable, actualizamos jQuery, mejoramos la documentación, y cambiamos Solr por ElasticSearch, e incluso se realizó un análisis comprando BOS antes y después del proyecto. Finalmente se desplegó el código en producción y se actualizo el código de los otros proyectos sin ninguna incidencia grave.

9.2. Posibles mejoras

Si pensamos en el recorrido del proyecto, podemos ver perfectamente como nos hemos dejado ciertas cosas en el tintero, si bien algunas de ellas son bastante prioritarias y es un riesgo no llevarlas a cabo:

- Más pruebas: es una realidad que ante la magnitud de un cambio como el realizado el riesgo de errores es bastante elevado, y a mayor cantidad de pruebas más sencilla es tanto la búsqueda de errores como su corrección, lo que disminuiría la probabilidad de fallos en el sistema y aumentaría la calidad del código. Hemos de entender que en un entorno empresarial se está muy controlado por los recursos disponibles, y no siempre es posible tener cubiertas todas las necesidades.

  A la hora de hablar de pruebas hemos de concretar que son necesarias pruebas unitarias, de integración y funcionales. Esto además reduciría el tiempo de prueba necesario por los QAs.

- Librerías: aunque todas las librerías fueron actualizadas sin problemas, es cierto que podríamos haber dedicado más tiempo a analizarlas en profundidad para analizar la verdadera necesidad que tenemos de ellas, pues puede ser que existan algunas nuevas librerías mas potentes o sencillas de usar a las que habría sido conveniente cambiar.

- Nuevas funcionalidades: vimos también como las nuevas versiones añadían bastantes funcionalidades, muchas de las cuales habría sido interesante estudiar su posible uso, pero que por falta de tiempo o verdadera necesidad fue imposible hacer.

- Documentación: la documentación es la gran olvidada en muchos proyectos -incluso algunas metodologías ágiles la rechazan- y, si bien es cierto que nosotros añadimos un buen sistema de documentación, podríamos haber dedicado más esfuerzos a documentar de manera mucho más extensa todas las interesantes novedades de cara al resto de la empresa.

9.3. Balance final

Haciendo balance del proyecto, podemos decir que ha permitido a la empresa mejoras en la seguridad del sistema gracias a la actualización de Django, aumento en el rendimiento en algunas partes de la aplicación, mayor legibilidad y extensibilidad de código gracias a la nueva estructura y a las pruebas, pues se puede hacer cambios con más seguridad de que nada esta fallando, además de la incorporación de muchas nuevas funcionalidades y facilidades para usar por los desarrolladores.

Es cierto que se retraso un poco el tener el sistema con Django 1.4.10 totalmente funcional y depurado, lo cual influyó en que se cancelase el actualizar Django a la más versión para más adelante en pos de otros desarrollos, pero el resultado final fue muy satisfactorio, y los trabajadores de la empresa quedaron muy contentos con los cambios hechos y las mejoras introducidas.

Si hemos de decir que factores han afectado tanto positiva como negativamente, diremos que el hecho del escaso conocimiento inicial de Python y Django por un lado, y de BOS por otro, hicieron que el rendimiento no fuese el de un trabajador experimentado, además de ser la primera vez que yo trabajaba en una empresa y además usando de manera totalmente real Scrum; y también fueron muchas las herramientas y procedimientos que tuve que aprender en poco tiempo.

Un aspecto positivo fue que se realizaron bastantes análisis en un comienzo que nos fueron muy útiles posteriormente, y si he de destacar algo por encima de todo lo demás, la introducción tanto de un sistema de integración continua como de un equipo de QA que estuviese al tanto de los cambios realizados por el equipo y que controlase que tanto las distintas funcionalidades como los tests fuesen hechos de manera correcta fue un factor totalmente fundamental para el éxito de este proyecto.

Aprovechamos para mencionar que siempre, y más aún cuando trabajamos en una

empresa, el hecho de saber coordinarte bien con el equipo y con los -muchas veces menospreciados- miembros de QA es mucho más importante que la experiencia, los conocimientos, o el presupuesto del que se disponga.

Referencias

[1] A. Holovaty and J. Kaplan-Moss, *"La guía definitiva de Django, 2010.* ANAYA.

[2] J. Watkins and S. Mills, *Testing It - An Off-the-Shelf Software Testing Process.* Cambridge University Press.

[3] M. D. Archer, *Getting Started in Currency Trading, Fourth Edition.* Hoboken, New Jersey: Jonh Wiley and Sons, Inc.

[4] G. G. Schulmeyer, *Handbook of Software Quality Assurance, Fourth Edition.* Artech House.

[5] B. Hill, J. Bacon, C. Burger, J. Jesse, and I. Krstic, *The Official Ubuntu Book (7th Edition).* Prentice Hall.

[6] M. Lutz and D. Ascher, *Learning Python, 5th Edition.* O'Reilly.

[7] R. Obe, *Postgresql: Up and Running, 2012.* O'Reilly.

[8] M. M. Nayrolles, *Mastering Apache Solr: A practical guide to get to grips with Apache Solr.* Paperback.

[9] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide.* O'Reilly Media.

[10] J. L. Carlson, *Redis in Action.*

[11] A. Álvarez García, R. de las Heras del Dedo, and C. L. Gómez, *Métodos Ágiles y Scrum.* ANAYA.

[12] H. J. W. Percival, *Test-Driven Development with Python.* O'Reilly Media.

[13] Chromatic, *Extreme Programming Pocket Guide.* O'Reilly Media.

[14] G. Unmesh, *Selenium Testing Tools Cookbook.* Paperback.

[15] T. A. Limoncelli, C. J. Hogan, and S. R. Chalup, *The Practice of System and Network Administration, Second Edition.* Addison-Wesley.

[16] B. O'Sullivan, *Mercurial: The Definitive Guide.* O'Reilly Media.

[17] J. F. Smart, *Jenkins: The Definitive Guide.* O'Reilly Media.

[18] A. Begel and N. Nagappan, *Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study.* Microsoft.

[19] H. Kniberg, *Scrum Y XP Desde Las Trincheras.* InfoQ.

[20] P. Galindo, M. Ángel López, and P. Lagares, *Universidad de Cádiz.* InfoQ.

[21] G. A. Campbell and P. P. Papapetrou, *SonarQube in Action.* Manning Publications Co.

[22] M. B. Doar, *Practical JIRA Administration.* O'Reilly Media.

[23] J. Chaffer and K. Swedberg, *Learning jQuery - Fourth Edition.* Paperback.

Apéndice A. Documentos adjuntos

ANEXO 1

En este primer anexo podemos ver el informe que nos proporcionó SonarQube sobre distintos detalles del código de BOS, como son el numero de líneas, funciones y clases, como se reparte el código entre las diferentes aplicaciones, la complejidad o la cantidad de comentarios

Initial Quality Analysis

BOS Django Update [BDU]

Summary

BOS Status at December 4, 2013.

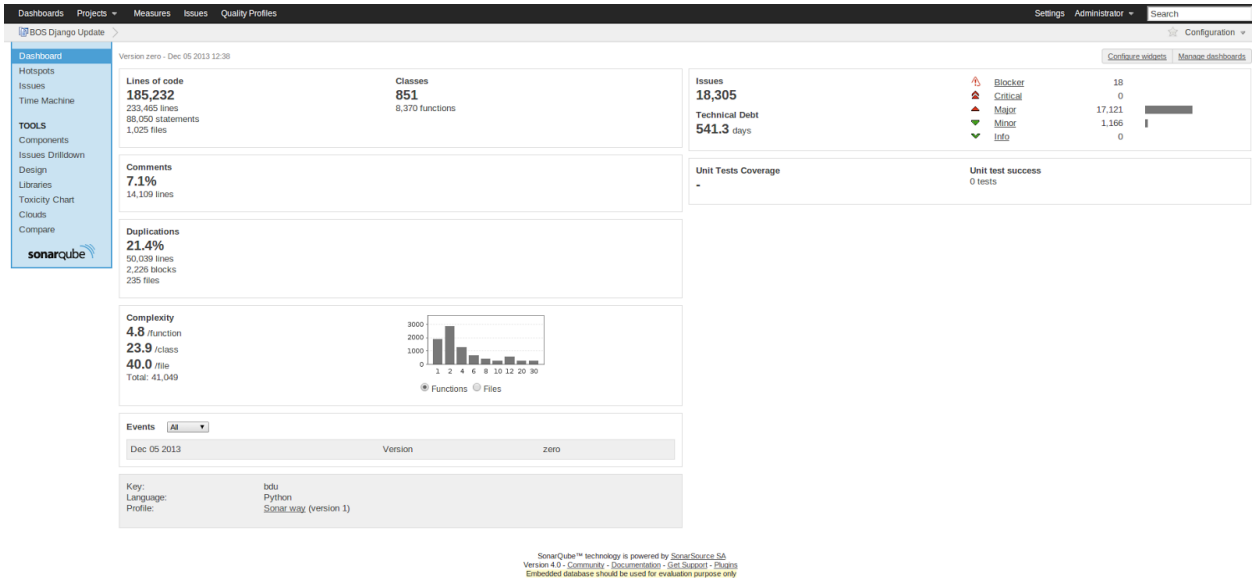
Status

Metric	Value
Lines	233.465
Lines of Code	185.232
Files	1.025
Clases	851
Functions	8.370
Python Lines	78.837

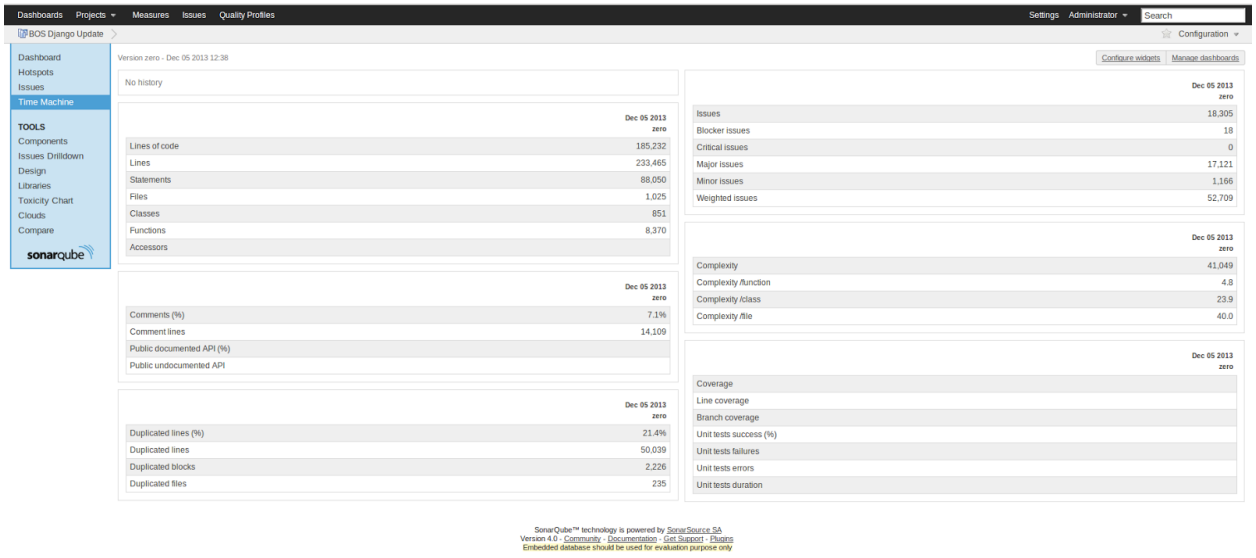
Quality

Metric	Value
% Comments	7,1%
Duplications	**50.039** (21,4%)
Global Complexity	**41,049** (High Risk)
Function Complexity	4,8 (Low Risk)
Class Complexity	**23,9** (High Risk)
File Complexity	**40,0** (High Risk)
Python Complexity	5,9 (Low Risk)

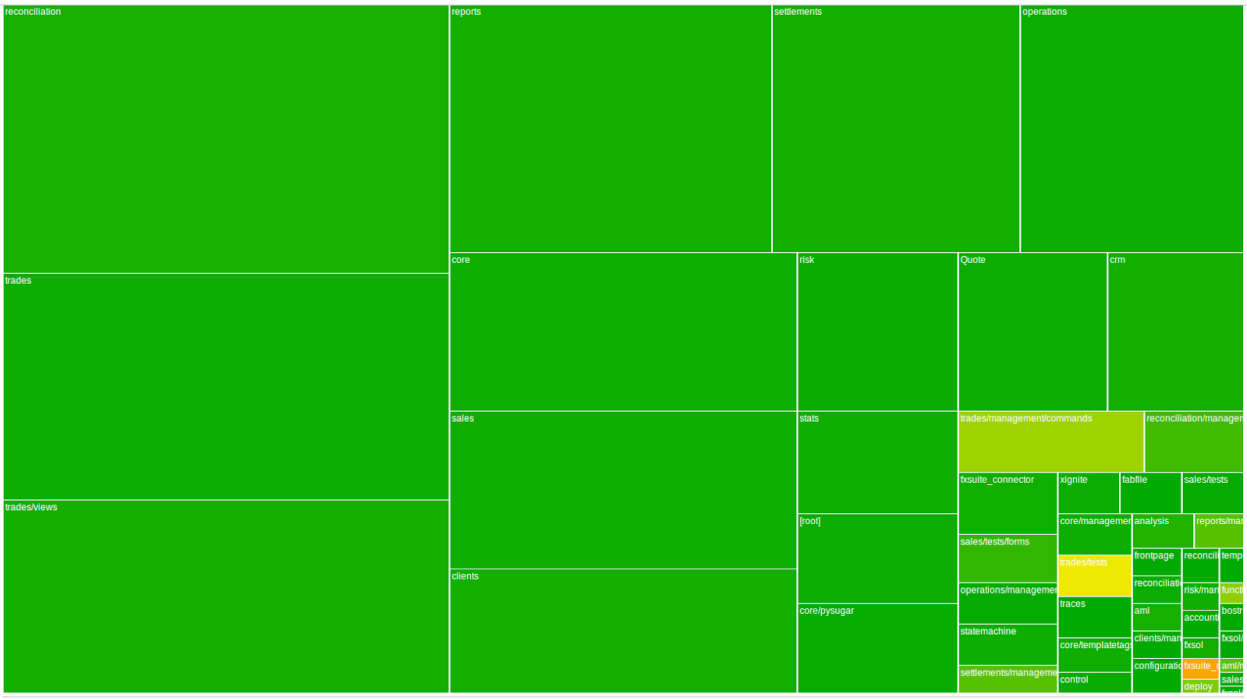
General Status



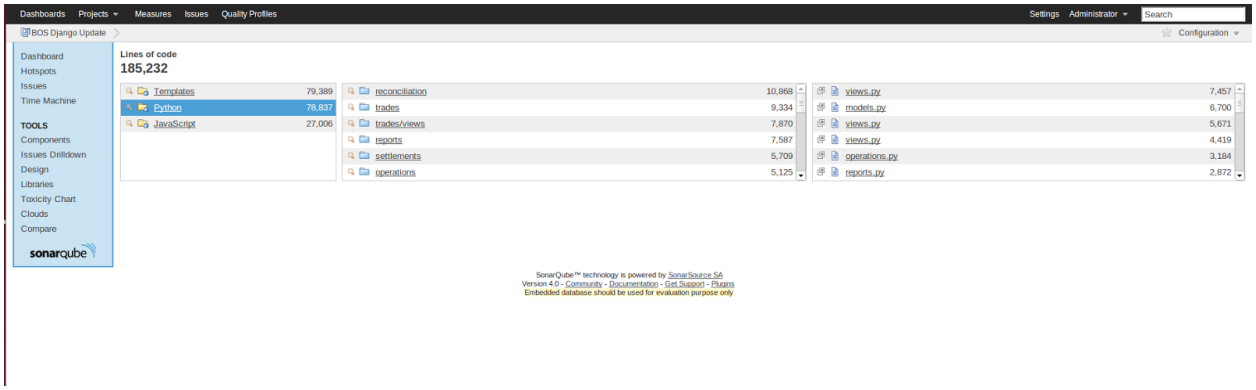
Initial Time Machine



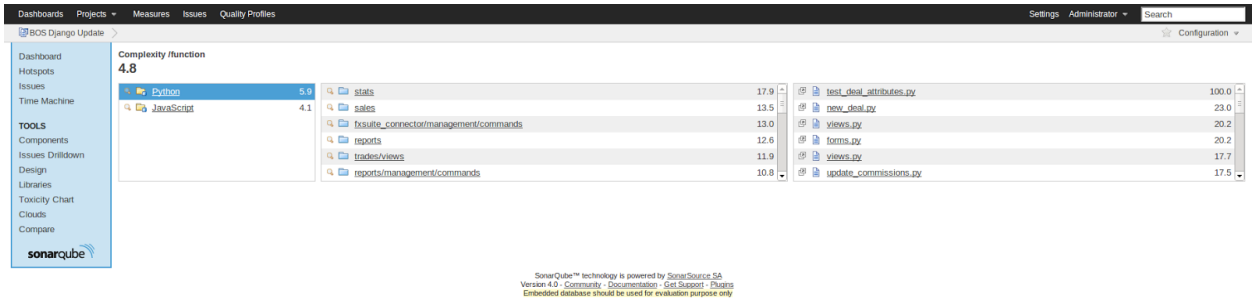
Code Distribution



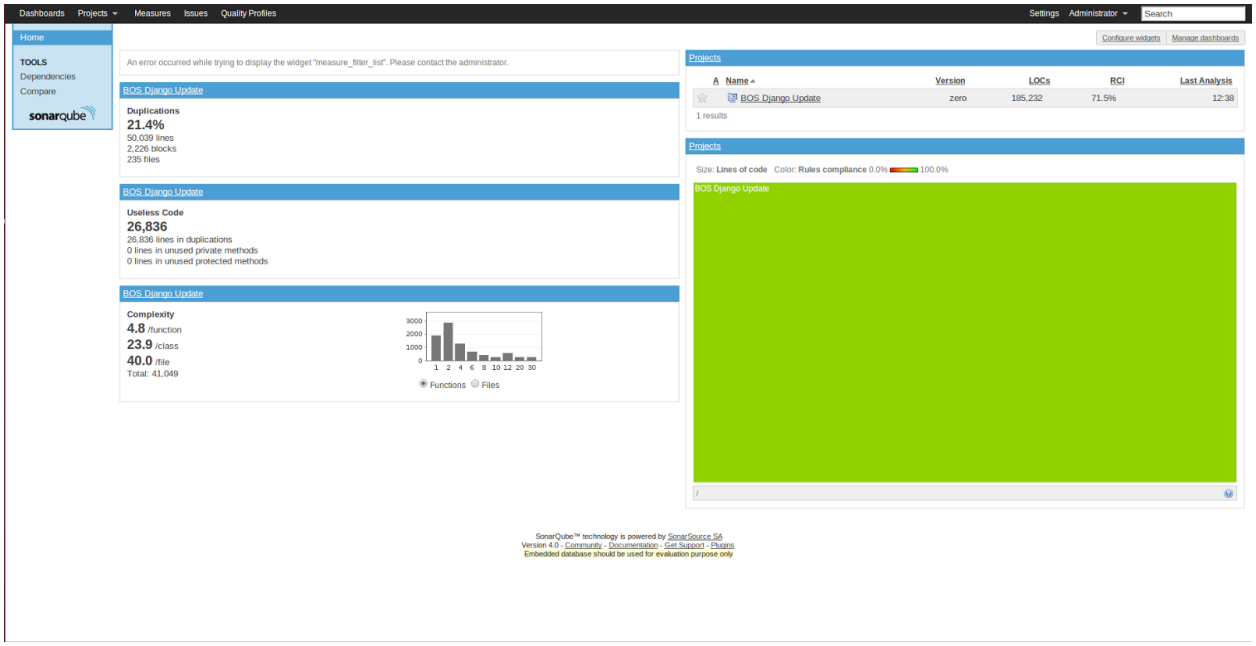
Lines of Code (LOC)



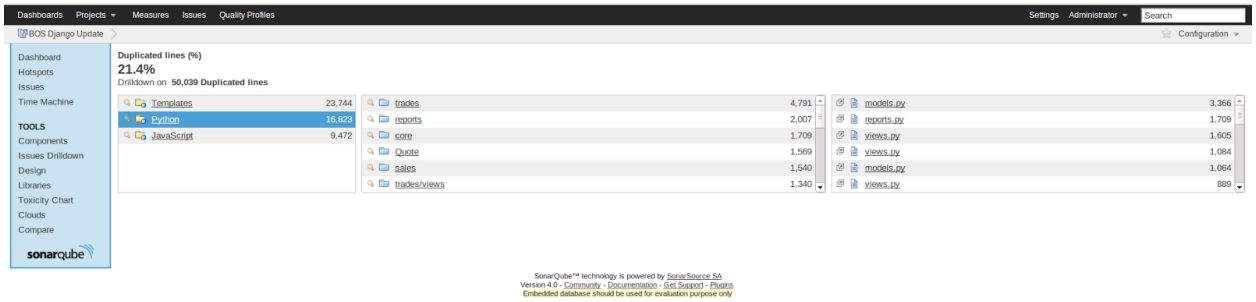
Complexity



Useless Code



Duplicated Lines



Legend

Legend of Cyclomatic Complexity:

Value	**Meaning**
1-10	Low Risk
11-20	Moderated Risk
21-50	High Risk
>50	Highest Risk (No testable program)

ANEXO 2

En este segundo anexo podemos ver un informe comparando el rendimiento de BOS antes y después de la actualización.

Detalle de las transacciones

Grado de mejora alcanzado por la aplicación, comparando el porcentaje de error de cada 'request' utilizada y su tiempo

Nombre del Caso de Prueba			Mejora de error	Mejora de respuesta (ms)	Observaciones
New Deal (10%)	Login	/accounts/login	0,00%	-150	
		/	0,00%	16	
	New Deal	/sales/	0,00%	57	
		/sales/new_deal	0,00%	279	
		/sales/new_deal/${ID_TRADE}/spot/receipt/html	0,00%	2854	
		/operations/allocations/funds_in	**100,00%**	**33145**	
Operations (70%)	Login	/accounts/login	0,00%	47	
		/	0,00%	-38	
	Pre-Execution	**/operations/allocations/pre_execution**	**0,00%**	**-36603**	
	Funds In Prending Auth	**/operations/allocations/funds_in_entries**	**0,00%**	**-6510**	
	Beneficiaries	/clients/beneficiaries/all/	0,00%	-796	
	Tradelog	/operations/tradelog	0,00%	-3774	
	Hold Queue	**/operations/allocations/hold_queue**	**0,00%**	**9761**	
	Payment	/operations/payments/all/	0,00%	3722	
	Clients	/operations/clients/	0,00%	2542	
	Open trade	**/operations/deal/client/show/30765/**	**0,00%**	**-7917**	
	Third Party	/operations/thirdparty/	0,00%	-74	
	Stats	/stats/	0,00%	249	
Risk (20%)	Login	/accounts/login	0,00%	-190	
		/	0,00%	-2	
	Open Position	/risk/open_position	0,00%	22	
	Credit Conditions	/risk/creditconditions/	0,00%	1734	

Informe de Evaluación del Rendimiento

Proyecto		Módulo		Versión
Django Update		Development2		Drosera
Comentarios sobre el resultado obtenido	Se comprueba una mejora del rendimiento en general, aunque hay resultados desfavorables, están influidos por el aumento de la base de datos, pero se puede comprobar que en la redirección a "Funds In" tras realizar un trade hay una mejora muy significativa en el tiempo medio de respuesta. Para próximas pruebas de rendimiento lo ideal sería tener las BBDD con un volumen parecido y los entornos completamente estables, es decir, mismo número de procesos en ejecución, tamaño de disco duro ocupado, etc. para que las pruebas sean totalmente fiables.			
Navegación	Navegación Conjunta	**Fechas de la prueba**	**Fecha inicio**	19/03/14
			Fecha fin	21/03/14
Parámetros utilizados para la prueba				
CPU de la máquina	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz			
Memoria RAM de la máquina	4 GB			
Métricas observadas				
Tiempo medio de respuesta para 10 usuarios (ms)				
Velocidad de respuesta mejorada del aplicativo	0			
Porcentaje de Errores encontrados para 10 usuarios (%)				
Grado de porcentaje mejorado alcanzado en la aplicación, comparando el porcentaje de error de ambas pruebas	0,89%			
Descripción de la Navegación				
Tiempo de respuesta permitido para las peticiones (aserciones de tiempo)	20.000 – 60.000 ms			
Periodo de tiempo en el que se mantendrá el número de usuarios concurrentes anunciado	3 minutos - 7 minutos			
Ritmo Transaccional para cada petición	**Tipo de Temporizador**	**Retardo**	**Desviación**	
Peticiones de Logueo	Gaussiano	2000 ms	500 ms	
Otras peticiones (búsquedas, edición, etc.)	Gaussiano	1000 ms	500 ms	