Análisis y arquitectura de herramienta Dashboard para HP Inc.

Muñiz Peralvarez, Manuel

Resumen– Este trabajo de final de carrera ha sido desarrollado con el fin de poner en práctica en un entorno empresarial real, una solución software relacionada directamente con el diseño web y la arquitectura software.

El propósito principal de este trabajo de final de carrera es elaborar una arquitectura sólida que solvente las complicaciones que ha encontrado HP Inc. con un sistema Dashboard que tiene funcionando actualmente.

En este proyecto se planea hacer un análisis de la tecnología que está en funcionamiento y, una vez conseguido esto, diseñar e implementar de una manera sólida una nueva arquitectura que consiga adaptarse a los requisitos que hayan solicitados las diferentes partes implicadas del proyecto.

Palabras clave– Dashboard, SWQA, AGILE, HTML5, CSS, Python, JS, GLOB, BS4, Plotly, ALM, Jpg, Png, XML, JSON, Parser, CRON, VLAN, arquitectura cliente-servidor, Api, Frontend, Backend, Template, Plugin

Abstract– This final degree project has been developed in order to put into practice in a material business environment, a software solution directly related to web design and software architecture.

The main purpose of this final degree project is to develop a solid architecture that solves the complications that HP Inc. has encountered with a Dashboard system that is currently operating.

In this project it is planned to make an analysis of the technology that is in operation and once this is achieved, to design and implement in a solid way a new architecture that manages to adapt to the requirements requested by the different parties involved in the project.

Keywords– Dashboard, SWQA, AGILE, HTML5, CSS, Python, JS, GLOB, BS4, Plotly, ALM, Jpg, Png, XML, JSON, Parser, CRON, VLAN, client-server architecture, Api, Frontend, Backend, Template, Plugin

1 Introducción y estado del arte

Hace años, HP Inc. decidió desarrollar un proyecto para mostrar diferentes informaciones en las pantallas que tenían en su oficina de una manera rápida para facilitar a sus trabajadores el poder ver el estado de los diferentes proyectos que se estaban llevando a cabo.

El sistema que se había implantado era una solución de tipo web que se generaba sobre un HTML de forma estática, es decir, existían unos procesos Backend que se encargaban de recoger toda la información que se quería mostrar en las pantallas con un fichero JSON y, seguidamente, el Frontend estático se encargaba de crear y dibujar las gráficas con los datos recogidos del fichero en una página web fija que mostraba el contenido en pantalla.

Este proyecto fue iniciado por el departamento de Software Warranty Quality Assurance el cuál, hace alrededor de un año, decidió intentar generar valor en las oficinas con las muestras de los datos recogidos de cada gran proyecto iniciado en la compañía.

Entre las diferentes limitaciones que podemos observar en este sistema sin entrar en detalle encontramos:

- Poca modularibilidad.
- Obligación de rehacer páginas nuevas continuamente desde 0.
- Sistema limitado y complicado de entender dado que su estructuración es nula (encontramos la mayoría de los scripts en un mismo fichero).

• E-mail de contacte: manel.mp.1998@gmail.com
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Yolanda Benítez Fernández
• Curs 2019/2020

- Utilización de lenguajes que actualmente se encuentran obsoletos y sin soporte nativo (Visual Basic, por ejemplo, utilizado para realizar queries en el Backend de la arquitectura.
- Estructura fija que complica el mantenimiento del código.

Fig. 1: Diseño estático previo del equipo SWQA de HP Inc.

Con el diseño e implementación de esta nueva arquitectura Dashboard que se ha diseñado, se planea estudiar a fondo las tecnologías utilizadas en el sistema implementado actualmente y, a su vez, hacer un estudio de las tecnologías que se utilizan hoy día para así poder seleccionar las que sean más apropiadas para el nuevo diseño.

Finalmente, se espera profundizar en los lenguajes de programación de diseño web Frontend y Backend y estudiar la arquitectura cliente-servidor de forma directa.

2 Objetivos

Como ya hemos comentado anteriormente, los objetivos principales del proyecto se han centrado en proporcionar una nueva arquitectura con base ampliable para un futuro que permita añadir contenido al Dashboard sin la necesidad de rehacer la página HTML entera de nuevo.

Se ha creído conveniente que la mejor manera de definir los objetivos es mediante una enumeración para indicar de manera resumida hasta dónde queremos llegar y un desglose diferenciando los que son prioritarios de los que no lo son:

Prioritarios:

1. Establecer una base para la nueva arquitectura. La idea principal de este proyecto es definir y realizar la principal línea de desarrollo software que solventará el problema al que se enfrenta actualmente HP Inc.

2. Estudiar la solución que se encuentra funcionando actualmente para valorar el cambio y margen de mejora disponible dentro de un alcance limitado por los recursos de la propia empresa sobre el proyecto e, intentar dar una solución óptima que permita mejorar el máximo posible la que se está utilizando ahora mismo.

3. Estudiar y aprender a proyectar la captura de requisitos a través de diferentes reuniones con las principales partes implicadas de un proyecto informático sabiendo recoger y ordenarlos de manera adecuada.

4. Aprender a trabajar sobre un proyecto real en equipo y definir su camino de desarrollo.

5. Estudiar alguna de las tecnologías hoy día utilizadas en el desarrollo web y ponerlas en práctica bajo un entorno de desarrollo real. Una de las principales tecnologías que se planea trabajar es CRON, JavaScript y principalmente la arquitectura cliente – servidor.

No prioritarios:

1. Implementar dicha arquitectura y ponerla en funcionamiento como una primera línea base de desarrollo del proyecto.

2. Aprender a trabajar juntamente con Apis en sistemas Backend para recoger todo tipo de información de diferentes fuentes y poder procesarlas seguidamente en un sistema informático.

3 Metodología utilizada

El proyecto se ha trabajado de forma iterativa sobre la metodología AGILE SCRUM.

Cada 3 semanas se han realizado reuniones periódicas de inicio de Sprint con el Product Owner del proyecto (que en este caso es el propio mánager del equipo) en las cuáles se ha preparado el lapso de trabajo que se inicia ese mismo día y se han discutido los problemas o apuntes encontrados en el Sprint anterior.

Pese a todas las inconveniencias que hemos tenido a causa de la pandemia del COVID19, la realización del TFG se ha acabado desarrollando satisfactoriamente con la única diferencia de que, esta vez, se ha trabajado de forma telemática con todas las herramientas que dispone HP y, se han realizado así, las reuniones de planificación a través de la herramienta Teams de Microsoft.

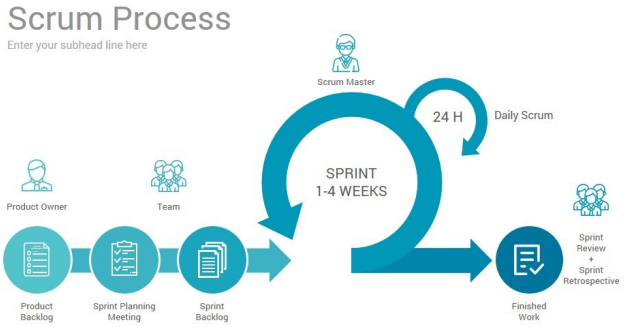

Fig. 2: Esquema gráfico del proceso SCRUM en metodología AGILE.

Además, cada 2 semanas se ha realizado una reunión personal de retrospectiva del proyecto para fomentar las discusiones acerca del mismo, poder solventar contratiempos que hubiesen aparecido a mitad de sprint y dar un feedback general de la evolución de éste y, poder tratar cualquier tema que pudiera surgir.

4 PLANIFICACIÓN

Para la organización y distribución del trabajo se ha decidido establecer una evolución de la planificación del desarrollo de la arquitectura a bajo nivel a modo de “Stories” que consisten básicamente en, una agrupación de pequeñas tareas englobadas en un solo enunciado dado que esta ha sido la forma de trabajar en Jira que se ha adoptado a lo largo de todo el proyecto.

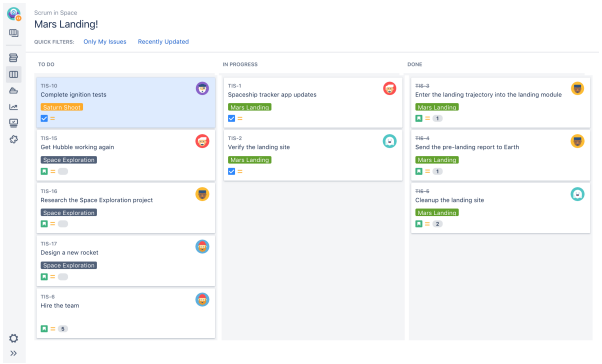

Fig. 3: Ejemplo de la distribución de las tareas sobre JIRA.

Cabe destacar que cada bloque de tareas depende totalmente del anterior y, por ende, suponemos su dependencia como total con la única diferencia del apartado de testing y documentación, que suponemos que se ha realizado a lo largo de todo el proyecto.

4.1. Estudio y análisis del sistema actual

En este apartado se ha incluído cada tarea realizada inicialmente para estudiar el sistema anterior.

Dentro de estas tareas encontramos el estudio previo del sistema anterior, diferentes actividades de investigación realizadas y pruebas realizadas con diferentes frameworks para estudiar su viabilidad:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- First search of the different frameworks to work with graphs.	-	20/1/2020
2- Research about out problems with google charts.	-	20/1/2020
3- Test the differents frameworks to prove his efficiency.	2	29/1/2020
4- Research alternatives for graphs.	1	29/1/2020
5- Study Bladimir actual working Dashboard.	-	20/1/2020
6- Talk with the responsible of the project about its future.	-	20/1/2020

4.2. Estudio de la problemática

En este apartado se han incluído las tareas que han tenido que ver con la valoración de los problemas encontrados en el Dashboard anterior realizado por SWQA así como un análisis del inicio del proyecto y su camino a seguir:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Study and search about the main problems of the SWQA architecture.	-	2/2/2020
2- Study documentation about the operation of google charts.1- Study and search about the main problems of the SWQA architecture.	-	2/2/2020
3- Study the backend Dashboard old process (Visual Basic).	-	2/2/2020

4.3. Captación de requisitos

En este apartado se ha incluído cada tarea que ha tenido que ver con la recolección de los diferentes requisitos de la nueva arquitectura así cómo las diferentes reuniones que se han llevado a cabo con los distintos responsables del proyecto:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Talk with the differents 3D managers to get information about the new Dashboard architecture.	-	2/2/2020
2- Study the requirements that the architecture must satisfy.	-	2/2/2020
3- Study the client-server model to apply it on the new architecture.	-	2/2/2020
4- Create a requirement report with the gathered information.	1 and 2	8/2/2020

4.4. Definición del nuevo sistema

En este apartado se han incluído las tareas que han tenido que ver con la definición y concepción del nuevo sistema desarrollado, la creación del fichero de configuración que será comentado en el subapartado 5.1 y, la estructura básica de la arquitectura a realizar:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Define the .cfg model that it's going to be implemented on the backend.	-	17/2/2020
2- Define the frontend process structure.	-	17/2/2020
3- Define the complete backend process structure.	-	17/2/2020
4- Search about SAMBA.	-	17/2/2020
5- Define basic shared directory structure.	-	17/2/2020

4.5. Implementación de la nueva arquitectura

En este apartado se han incluído las tareas que han tenido que ver con la implementación de la nueva arquitectura tanto Frontend, Backend, cómo los procesos que se encargan de gestionar y actualizar activamente los datos de los plugins entre máquinas, la implementación del sistema que se encarga de comunicar los diferentes dispositivos y el estudio sobre cómo tratar los datos recogidos por las diferentes apis:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Make a first backend script to process a simple .cfg user file.	-	30/3/2020
2- Implement frontend process structure.	-	30/3/2020
3- Adapt SWQA model to the new frontend process model.	2	5/4/2020
4- Create different CSS templates to use it on the new frontend display model.	-	30/3/2020
5- Pick up the information collected via backend (JSON format) and collect it properly on frontend model.	2	5/4/2020
6- Implement complete backend process.	-	30/3/2020
7- Implement the shared folder architecture with SAMBA and test it.	6	5/4/2020

4.6. Plugins API

En este apartado se han incluído las tareas que han tenido que ver directamente con el desarrollo de los diferentes plugins pensados para la arquitectura:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Implement a First version of the SonarQube plugin.	-	13/4/2020
2- Implement a First version of the Jira plugin.	-	13/4/2020
3- Process the JSON information collected on apis correctly on frontend.	1 and 2	30/4/2020
4- Study and implement a cache for the backend process API libraries (to avoid making same requests to the server).	1 and 2	30/4/2020
5- Implements Plotly charts for the Jira API.	2	30/4/2020

4.7. Tareas opcionales

En este apartado se han incluído las tareas que no fueron pensadas desde un principio a ser desarrolladas pero que se han acabado realizando dentro del proceso de desarrollo de la arquitectura:

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Implement the SWQA process to the new frontend architecture.	-	30/4/2020
2- Develop an automatic script deployment process to update the project.	-	30/4/2020
3- Study the possibility of parallelizing the backend process.	-	30/4/2020 (Solution studied but postponed to the future).

4.8. Testing

En este apartado se han incluído todas las tareas que han tenido que ver con el testing de la arquitectura desarrollada.

El testing funcional de la arquitectura queda fuera del alcance del trabajo TFG propuesto dado que se centra en el diseño de la arquitectura y será realizado de forma continuada por diversos responsables del equipo especializados en el ámbito.

Story (Tareas)	Dependencia entre fases	Fecha estimada
1- Low level testing (Backend)	-	iterative (done with all the team)
2- Low level testing (Frontend)	-	iterative (done with all the team)
3- Low level testing (Jira library)	-	iterative (done with all the team)
4- Low level testing (SonarQube)	-	iterative (done with all the team)

5 Arquitectura propuesta: análisis y diseño

La existencia de este proyecto viene determinada por la necesidad de poder hacer el anterior Dashboard configurable para así evitar programar cada cierto tiempo toda una página HTML con información diferente y, por ello, el principal objetivo en el que centrarse es proponer una solución para esto.

A continuación, haremos una definición y un pequeño análisis de lo que ha suponido la arquitectura finalmente y veremos cómo se ha tratado el flujo de los datos a través de los diferentes componentes que forman el sistema.

5.1. Vista general del diseño

Ante la imposibilidad de poder generar una solución totalmente dinámica que se encargue de darle la posibilidad al usuario de elegir exactamente dónde y cómo colocar un plugin en pantalla (debido a que sería una inversión demasiado costosa y se extendería en tiempo y recursos de la empresa), la solución más viable pensada es la que comentaremos a continuación y se explica de forma gráfica en la estructura jerárquica gráfica mostrada en el subapartado A.1 del anexo del documento.

Adhiriéndonos a las restricciones y requisitos recolectados, la idea base es tener un fichero de configuración que divida la información en una estructura de fichero XML mediante el cuál, establezcamos una separación de cada página que vamos a querer mostrar sobre un cliente y, a su vez, cada página disponga de diferentes módulos que definan el tipo de contenido que se quiere mostrar.

Dado que la idea básica es permitirle al usuario cierta "libertad" de movimiento de los diferentes plugins del Dashboard, la solución conseguida es crear unos "templates" que funcionen cómo esqueleto del Dashboard en pantalla, es decir, unos recuadros sin contenido que no se visualizarán en el HTML pero que si nos servirán como localización para colocar la información. Estos templates funcionarán como el esqueleto que sustente cada página a renderizar y servirá para establecer la disposición de los contenidos que se generarán por cada módulo.

Para crear cada uno de los templates nos ayudaremos con la herramienta web LayoutIt la cuál nos proporcionará un código Grid puro sin contenido y nos facilitará la adaptación al modelo que queremos implementar en la arquitectura. La gran ventaja que tenemos con este modelo de diseño es que, cualquiera que quiera crear un Grid e implementarlo en el sistema, podrá hacerlo de una manera sencilla ahorrando en gran medida todos los costes que deberían asumirse en el caso de querer hacer algo realmente dinámico. Dado que en este proyecto nos centramos solo en lo realmente importante, la arquitectura general de todo el sistema, solamente han sido implementados dos tipos de módulos:

SixModule display: El módulo que hemos utilizado para adaptar el Dashboard existente en SWQA y permitir hacerlo modulable en el caso de querer cambiar un plugin por otro.

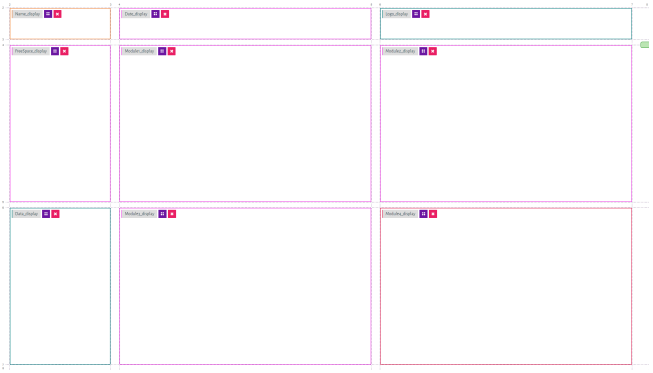

Fig. 4: Diseño CSS Grid visual del sistema creada con Layoutit del template SixModule.

SingleModule display: Este módulo ha sido diseñado principalmente para utilizarlo cuándo queramos mostrar algo que quiera mostrarse en toda la pantalla (un HTML, una imagen, un PowerPoint, etc.) conservando el banner original. Para mas información acerca de las posibilidades con este módulo consultar el Manual de Usuario que se encuentra en la documentación completa del proyecto.

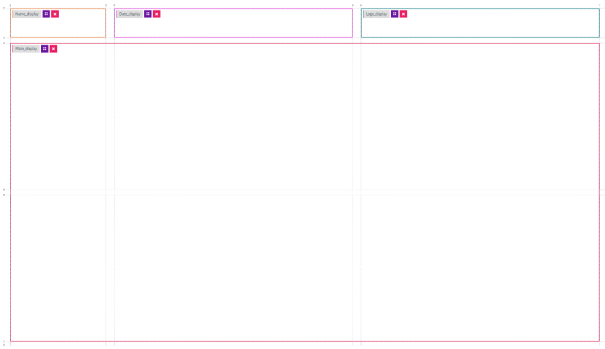

Fig. 5: Diseño CSS Grid visual del sistema creada con Layoutit del template SingleModule.

5.2. Vista general de la nueva arquitectura

Tal y como hemos comentado en el apartado número 6, los principales actores o roles del sistema son: 1-n usuarios, 1-n clientes y 1 sistema servidor.

(El diagrama gráfico explicativo de la arquitectura diseñada para soportar al nuevo sistema Dashboard puede encontrarse en el subapartado A.2 del anexo adherido al final del documento)

El flujo de trabajo de actualización normal presuponiendo que el fichero de configuración introducido por el usuario es correcto es el siguiente:

1. El usuario rellenará un fichero con formato .cfg utilizando el lenguaje de marcado XML especificando la configuración del sistema.

2. El usuario colocará dentro de la carpeta clients/ los diferentes archivos de configuración .cfg que quiera que sean procesados dependiendo de los clientes que se encuentren funcionando (dashboard1.cfg, dashboard2.cfg, etc.)

3. En el caso de querer incluir algún archivo extra en el sistema que necesiten todos los clientes procederá a hacerlo en el directorio common/ (ficheros HTML compartidos, videos, imágenes, etc.)

4. Cada cierto intervalo de tiempo especificado por la Workstation mediante la herramienta CRON, el sistema lanzará de forma automática el script que se encargue de leer todos los ficheros .cfg que haya introducidos en el sistema y se encargue de procesarlos para devolver un fichero JSON (que será el que recoja después el cliente de forma periódica) con toda la información necesaria (datos recogidos de las diferentes Apis incluidos) para mostrar el Frontend en el cliente.

5. El cliente, mediante un CRON, se encargará de descargar automáticamente cada cierto tiempo la información recogida en formato JSON procesada por el sistema y todos los archivos que se encuentren en la carpeta common/ y procederá a sobrescribir los ya existentes por los nuevos.

6. Al estar configurado el cliente para lanzar el script al iniciarse la Raspberry Pi 3 (y encontrarse, podemos decir, funcionando de manera "permanente"), automáticamente se mostrarán en pantalla los nuevos datos recogidos por el JSON.

(El flujo de trabajo de la arquitectura puede encontrarse en el subapartado A.3 del anexo adherido al final del documento)

5.3. Diseño sistema Backend

(El diagrama de flujo del sistema backend se puede encontrar en el anexo adherido al final del documento en el subapartado A.4).

El flujo principal del contenido backend es el siguiente: (Este proceso se repite tantas veces como ficheros haya para procesar en el directorio clients/)

1. El proceso Backend se encarga de coger el fichero .cfg y leerlo mediante la librería Glob de Python.

2. Seguidamente, el parser BeatifulSoup leerá el fichero xml y lo anotará como xmlDom para procesarlo.

3. El primer paso es procesar el clientId y el footerText y, seguidamente, se llamará al método processClientPages(xmlDom) que es el que se encarga de procesar el resto del fichero de configuración.

4. Este método encapsula el procesado de cada página del fichero de configuración y dentro de esta, engloba el procesado de los módulos y sus respectivas configuraciones.

5. Una vez se ha procesado todo el fichero de configuración, cada parte del fichero de configuración queda almacenada en una variable temporal que contiene el datos a incluir en el fichero JSON final y, por ende, el último paso consiste en precisamente eso: se hace un writeOutputFile(clientId) y se escribe en un archivo JSON de salida para que pueda tratarlo seguidamente el Frontend cuando recoja la información.

6. El proceso leerá el siguiente fichero de configuración y volverá al paso número 1.

5.4. Diseño sistema Frontend

(El diagrama de flujo del sistema frontend se puede encontrar en el anexo adherido al final del documento en el subapartado A.5).

El flujo principal del contenido frontend es el siguiente:

1. El sistema recogerá el JSON creado en el proceso Backend anterior y lo utilizará como banco de datos y configuración principal de lo que va a mostrar en pantalla.

2. Se establecerá un contador con frameTimeoutArray y el intervalo que va a durar en pantalla la página n (recogido en el archivo JSON del fichero de configuración .cfg)

3. Se lanza el método switchProgram() que funciona como método iterativo principal y se encarga de pintar todo lo que hay en pantalla. Este método a su vez dispondrá de un frameIndex, un contador que nos indicará en que página estamos y a su vez se colocará a 0 al llegar a la última página para comenzar por la primera de nuevo (y conseguir así un ciclo automático iterativo)

4. drawPageFunctions() se encarga de pintar el tipo de pantalla que queremos mostrar (tal y como hemos explicado antes en el apartado 7.1, son unos templates que funcionan como estructura básica de la página para permitir que sea modulable).

5. A partir de aquí processModules() se encargará de pintar cada módulo que hemos seleccionado uno detrás de otro.

6. Cuando el intervalo de tiempo que la página debe durar en pantalla llegue a su final, ésta cambiará a la siguiente y volveremos a iterar sobre el paso número 3.

6 RESULTADOS DE LOS PLUGINS REALIZADOS

Pese a que este trabajo se centra principalmente en la realización de la arquitectura 3D Dashboard elaborada e implementada a lo largo de los meses, me gustaría hacer un inciso y sacar algunas conclusiones de algunos de los plugins de la arquitectura para ver como han ido de forma general.

Para no repetir de forma gráfica todos los plugins realizados en este proyecto, me gustaría comentar que se pueden encontrar ordenados en el manual de usuario adherido a la documentación del proyecto.

Si nos fijamos en cada uno de ellos podremos ver que, en el caso de Sonarqube, han quedado unos plugins simples (dado que se limita a mostrar datos en una tabla) pero totalmente funcionales y ajustados a lo que pedía desde un principio obtener mi Product Owner de esta API.

En el caso de Jira se han acabado realizando menos plugins a los previstos a principio del proyecto dado a una conversión de la herramienta utilizada para gestionar antiguamente (la cual no me es posible comentar derechos reservados de la empresa) por la metodología ágil de la empresa y su traspaso a Jira (movimiento de la empresa que no había sido previsto al principio del proyecto).

Si observamos la *Fig.1* del artículo, podremos observar tal y cómo he comentado que los siguientes plugins han desaparecido:

- Found and fixed: su desaparición viene causada dado que en Jira su gestión de valores de las tareas es distinto.
- Open defects log: su desaparición viene causada dado que era una gráfica que, según expertos en SWQA de la empresa, no mostraba valores de interés para los empleados.
- El plugin que muestra el número de tareas en función de un RC: su desaparición viene causada dado que eran valores que ya aparecían en la tabla "Open defects by team" y, por lo tanto, era redundante.

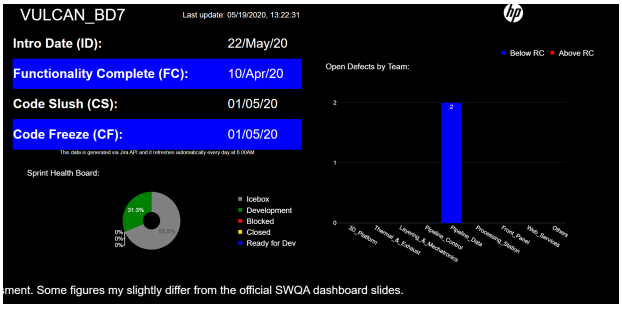

Fig. 6: Ejemplo de una plantilla de Dashboard actual de 3 módulos.

Finalmente, si nos fijamos en la documentación completa que podemos encontrar del proyecto, veremos que se han realizado otros tipos de plugins más simples que se limitan a permitir la inclusión de imágenes, vídeos, contenido de texto, etc. sobre los templates y, pese a que no se suelen utilizar de forma normal en el día a día de la empresa, cabe decir que es trabajo que está implementado y en cualquier momento podría usarse.

7 Conclusiones

Tal y como hemos podido comprobar, antes de ponernos a programar es necesario hacer un estudio de viabilidad y análisis inicial del proyecto. Un proyecto no deja de ser una planificación en la que se encuentran un conjunto de actividades a realizar de forma interrelacionada y, por ello, debemos estudiar qué relación existe entre las tareas para prevenir bloqueos y no frenar el flujo de trabajo.

Muchas de las tareas implicadas dentro de esta nueva arquitectura han sufrido muchos "delays" o bloqueos ya que HP Inc. es una empresa de un volumen gigantesco y, para desarrollar según qué proyectos, se debe coordinar con los responsables superiores y conseguir una serie de permisos que se pueden demorar en el tiempo.

Además, hemos podido comprobar perfectamente que es totalmente necesario hacer una valoración de coste y personal dado que, el proyecto siempre debe tender a adaptarse a estas restricciones. Por tiempos de desarrollo, no se puede hacer un sistema 100 % dinámico que se adhiera a los requisitos "soñados" pero, lo que si podemos hacer es algo funcionalmente estable que rinda de forma parecida que abarate los costes y rinda de forma parecida.

Cómo seguramente se habrá observado, en este proyecto no se ha realizado una valoración de costes y personal de forma explícita pero lo que sí se ha hecho es definir unos requisitos que ya venían anclados a restricciones que delimitaban el alcance de este. Como conclusión es importante remarcar la importancia de establecer una buena definición de requisitos con el Product Owner del proyecto para poder desarrollar lo esperado y ajustarse al alcance marcado porque, sobrepasar o no llegar a esos límites siempre es una mala decisión que puede afectar de forma muy negativa al desarrollo general de la arquitectura.

7.1. Línea de trabajo futuro

El objetivo principal del proyecto ha quedado perfectamente suplido puesto que el principal problema a resolver era realizar una arquitectura sólida que solventara todos los problemas que fueron surgiendo con el viejo diseño y, a partir de ahí, establecer una base de desarrollo para futuros plugins.

Si observamos la documentación completa, podremos ver que actualmente se han realizado diferentes plugins:

- Plugins para insertar contenido HTML.
- Plugins para insertar fotos y vídeos.
- Plugins para mostrar información de SonarQube.
- Plugins para mostrar información de Jira.

Como adición al proyecto se podría considerar la creación de cualquier tipo de plugin que se imagine (y sea viable) y añadirlo funcionalmente a la arquitectura actual.

7.2. Puntos de mejora de la solución actual

Como primer punto de mejora, yo vería ideal estudiar una solución (a partir de la ya diseñada) que mirase de paralelizar los scripts que procesan los ficheros de configuración dado que actualmente se ejecutan de una forma lineal de una forma muy "metódica" (aunque destacar que 100 % funcional).

Además, como segundo punto de mejora podría mirarse de hacer un estudio en profundidad a todo el código y ver si se pudiese hacer algún "refactor" de éste para optimizarlo aún todo y que esto no es un punto crítico dado que ahora mismo ya funciona tal y cómo se esperaba.

Finalmente, cómo punto final de mejora de la solución actual, vería importante reconsiderar todos los diseños UI

de los plugins actuales dado que ahora mismo están considerados para que sean funcionales pero, se podría profundizar en realizar un trabajo estrechamente relacionado con la "user experience" para conseguir mejores diseños porque, cabe recordar, que éste es un proyecto que se está viendo durante todo el día en las oficinas de HP Inc. y, una de las principales metas a combinar a mi parecer en el diseño de plugins son las siguientes:

- Conseguir plugins que sean agradables visualmente.
- Conseguir plugins que muestren información relevante

7.3. Valoración final

Cómo puntos a mejorar en el desarrollo personal me gustaría recalcar que las primeras semanas de desarrollo no fueron fáciles dado que entraba en un ámbito parcialmente nuevo (dado que desconocía la mayoría de tecnologías), tuve que invertir mucho tiempo en aprender cómo funcionaban algunas de las estructuras y tecnologías internas de la empresa y, fue un camino que de primeras se me hizo algo cuesta arriba a pesar de que al final ha sido una experiencia gratificante.

Finalmente, comentar que a pesar de los tiempos que estamos viviendo, he aprendido una experiencia muy positiva dado que el trabajar desde casa no me ha frenado en ningún momento, sino que, gracias a disponer de todo mi material de trabajo he podido avanzar con mucha más soltura en este proyecto.

8 Agradecimientos

Me gustaría agradecer a toda mi familia el apoyo facilitado a lo largo de todo mi trayecto académico, a Carmelo García, mi tutor en prácticas, tutor en el desarrollo del TFG en la empresa y, compañero de trabajo en HP Inc., a Sebastian Ybarra, mánager principal de trabajo y, a Yolanda Benítez, mi tutora docente de TFG proporcionada por la Universidad Autónoma de Barcelona toda la ayuda que me han proporcionado en la realización de este proyecto.

Referencias

[1] LÓPEZ PEÑA ANTONIO M. (2019). Documentación recogida de la asignatura "Laboratorio Integrado de Software". Cataluña: Univeristat Autónoma de Barcelona.

[2] PONSA MUSSARRA D. (2019). Documentación recogida de la asignatura "Requisitos del Software". Cataluña: Univeristat Autónoma de Barcelona.

[3] LLADÓS CANET J. (2018). Documentación recogida de la asignatura "Ingeniería del Software". de la Univeristat Autónoma de Barcelona.

[4] AULI LLINAS F. (2019). Documentación recogida de la asignatura "Tecnologías para el desarrollo de internet y web". Cataluña: Univeristat Autónoma de Barcelona.

[5] CONSEJO DE COLEGIOS DE INGENIEROS DE INFORMÁTICA (2016). Plantilla básica para la elaboración de la documentación de proyectos de ingeniería informática. Artículo. CCII

[6] JEFF CHASE. Sockets and clients/server communication. Artículo. Durkham: Duke University.

[7] EQUIPO VÉRTICE (2009). Técnicas avanzadas de diseño web. España: Editorial Vértice.

[8] KRUG S. (2000). Don't make me think. United States: New Riders Press.

[9] CECIL MARTIN R. (2008). Clean code a handbook of agile software craftsmanship. United States: Prentice Hall.

[10] WOLF PROJECT. Project management basics. https://wolfproject.es/que-es-el-project-management/ [Consulta: 7 de enero 2020]

[11] BBVA. Metodología AGILE, la revolución de las formas de Trabajo. https://www.bbva.com/es/metodologia-agile-la-revolucion-las- formas-trabajo/ [Consulta: 10 de enero 2020]

[12] OBS BUSINESS. Que es AGILE y cuáles son los 12 principios de su modelo. https://obsbusiness.school/es/blog-project-management/ metodologias-agiles/que-es-agile-y-cuales-son-los-12- principios-de-su-modelo [Consulta: 10 de enero 2020]

[13] CIBERNOS. Metodologías AGILE, comparación. https://www.cibernos.com/blog/desarrollo-de-software/ metodologias-agile-cual-es-la-mejor [Consulta: 10 de enero 2020]

[14] PLATZI. Que es Frontend y Backend. https://platzi.com/blog/que-es-frontend-y-backend/ [Consulta: 17 enero 2020]

[15] MICROSOFT. Documentación Visual Basic. https://docs.microsoft.com/es-es/dotnet/visual-basic/ [Consulta: 20 enero 2020]

[16] PYTHON. Documentación Python. https://docs.python.org/3/ [Consulta: 25 de enero 2020]

[17] W3 SCHOOLS. Diseño, paradigmas web y documentación. https://www.w3schools.com/ [Consulta: 25 de enero 2020]

[18] BEATIFUL SOUP. Documentación bs4. https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [Consulta: 3 de febrero 2020]

[19] CURIOUS CONCEPT. JSON formatter. https://jsonformatter.curiousconcept.com/ [Consulta: 7 de febrero 2020]

[20] MOZILLA. Documentación JSON. https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos-globales/JSON [Consulta: 14 de febrero 2020]

[21] STACK OVERFLOW. Resolución de dudas y problemas. https://stackoverflow.com/ [Consulta: 30 de marzo 2020]

[22] PLOTLY. Documentación Plotly https://plotly.com/python/ [Consulta: 20 de marzo 2020]

[23] GITHUB. Código fuente Plotly https://github.com/plotly/plotly.py [Consulta: 20 de marzo 2020]

[24] ORCA. Setting up de Orca https://sites.google.com/site/orcainputlibrary /setting-up-orca [Consulta: 20 de marzo 2020]

[25] JIRA. The REST API developer documentation https://developer.atlassian.com/cloud/jira/ platform/rest/v3/ [Consulta: 15 de abril 2020]

[26] JIRA. General documentation https://developer.atlassian.com/cloud/jira/ platform/rest/v3/ [Consulta: 15 de abril 2020]