ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE

Curso Académico 2022/2023

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN SHINY PARA EL CONTROL ESTADÍSTICO DEL CONTENIDO EFECTIVO DE PRODUCTOS ENVASADOS

Autor: David Ángel Leo Acedo

Director: Emilio López Cano

1. RESUMEN

Entre las múltiples opciones para el manejo de datos, el paquete para *R, Shiny* despunta como una herramienta muy útil para el desarrollo rápido y sencillo de pequeñas aplicaciones web de visualización. Su simplicidad de uso, sin embargo, favorecen que muchas de las aplicaciones creadas con él tengan problemas para aumentar su complejidad, y mantener las mejores prácticas de la ingeniería del software en su desarrollo.

En este trabajo de fin de grado se creará una aplicación web usando el paquete Shiny para proporcionar a los usuarios una manera sencilla de aplicar los controles de calidad obligatorios recogidos en el Real Decreto 1801/2008, de 3 de noviembre, por el que se establecen normas relativas a las cantidades nominales para productos envasados y al control de su contenido efectivo.

En el proceso, se propondrá un flujo de trabajo ágil para el desarrollo de una aplicación *Shiny*, en línea con los estándares actuales de la Industria del Software y se estudiará como aplicar sobre esta clase de desarrollos el testing automatizado, la integración y el despliegue continuos.

Por último, se analizará que ofrece la comunidad Shiny para hacer de sus aplicaciones desarrollos más robustos, utilizando los paquetes y la filosofía de trabajo del *golemverse*.

PALABRAS CLAVE: *Aplicaciones web, R, Shiny, control de calidad, metrología, Estadística*

2. ÍNDICE

3. INTRODUCCIÓN

Desde su lanzamiento en 2012, *Shiny* se ha popularizado como una herramienta muy útil para realiza aplicaciones relacionadas con el análisis y la visualización de datos [1]. Gracias al uso extensivo de R como lenguaje para proyectos científicos y estadísticos, *Shiny* ha crecido para convertirse, no solo en una herramienta de prototipado, sino en un *framework* de desarrollo de pleno derecho, enfocado siempre en la simplicidad de uso. [2]

Debido a este enfoque en la simplicidad, sin embargo, las aplicaciones *Shiny* tienen limitaciones importantes en cuanto a complejidad, personalización, escalabilidad, mantenibilidad y seguridad.

Para abordar alguno de estos problemas, han surgido alrededor de *Shiny* una amplia variedad de paquetes adicionales. Sin embargo, este ecosistema vibrante añade una capa de complejidad al desarrollo de aplicaciones que, originalmente, estaban destinadas a ser desarrolladas por personas ajenas al diseño web.

Por otro lado, el Real Decreto 1801/2008, de 3 de noviembre, por el que se establecen normas relativas a las cantidades nominales para productos envasados y al control de su contenido efectivo es la norma española que indica a las empresas que controles de calidad mínimos deben realizar sobre sus productos envasados en cuanto al contenido de éstos. Este Real Decreto obliga a todas las empresas productoras o distribuidoras a realizar distintos análisis sobre los lotes que pretende comercializar dentro de la Unión Europea.

Pese a la gran cantidad de empresas obligadas a realizar estos análisis, no existe ninguna aplicación de código abierto que implemente estos procesos. Esto fuerza a pequeñas y medianas empresas a depender, o bien de costosas aplicaciones ERP, maquinaria especializada o en soluciones caseras sobre software propietario.

Es en este contexto en el que se realiza este trabajo. Nuestro objetivo es demostrar las capacidades de *Shiny* desarrollando una aplicación para gestionar los procesos del Real Decreto 1801/2008 y establecer una metodología de desarrollo sólida para las aplicaciones *Shiny*, utilizando los principios de la ingeniería del software actual, que equilibre la simplicidad de desarrollo de *Shiny* con las necesidades de la construcción de aplicaciones más complejas. Con esto, esperamos contribuir a una mejor y más eficiente práctica de desarrollo de aplicaciones *Shiny*.

Este trabajo será presentado en el XL Congreso Nacional de Estadística e Investigación Operativa, sesión del Grupo de Trabajo en Software y Computación en Estadística e Investigación Operativa.

3.1. EL PAQUETE SHINY

Shiny [3] es un paquete para R de código abierto que permite a los usuarios construir aplicaciones web interactivas desde R enfocadas principalmente a la creación de tableros de visualización (Ilustración 1) e interfaces de usuario para programas de procesamiento de datos. Usando funciones y objetos de R consigue generar el código

HTML, *CSS* y *JavaScript* necesario para construir una aplicación web y el servidor necesario para soportarla.

Para ello, además de la generación de código en base a componentes de alto nivel, *Shiny* se sirve de una implementación relativamente intuitiva de la programación reactiva, para procesar los inputs del usuario y generar los outputs requeridos.

Todo ello ha hecho de *Shiny* una forma muy popular entre analistas de datos, estadísticos y matemáticos, que pueden tener conocimientos de *R,* pero una experiencia limitada en programación web.

Su fama, la implicación de organizaciones de referencia y la contribución desinteresada de muchos desarrolladores alrededor del mundo ha propiciado que surjan alrededor de *Shiny* una gran variedad de paquetes auxiliares que proporcionan nuevas funcionalidades y ayudas a la hora de desarrollar una aplicación con él.

Sin embargo, todas estas facilidades conllevan una serie de limitaciones a la hora de desarrollar aplicaciones más grandes o con requisitos más estrictos. Algunas de ellas son:

1) Rendimiento y escalabilidad: El núcleo de R no admite paralelización, una característica importante que hace que otros lenguajes sean mucho más adecuados para soportar un servidor destinado a un público amplio. De hecho, una misma instancia de *Shiny* solo puede dar servicio a un número muy reducido de usuarios.
2) Personalización: Dado el alto grado de abstracción que ofrece *Shiny* sobre los elementos HTML que construye, la personalización nativa es limitada. Es posible dar estilos al código HTML autogenerado, aunque esto puede traer problemas de mantenibilidad por sí mismo.
3) Mantenibilidad: La gestión de la reactividad puede volverse bastante compleja rápidamente y las opciones de modularización de una aplicación son limitadas. Además, para construir elementos avanzados puede ser necesario incluir fragmentos de código *JavaScript* y *CSS* que no siempre se integran de forma limpia en el código R.
4) Persistencia: *Shiny* no ofrece la posibilidad de gestionar sesiones persistentes una vez el usuario cierra la página.
5) Seguridad: *Shiny* no ofrece opciones de autentificación ni control de acceso.

Muchas de las contribuciones de la comunidad mencionadas anteriormente intentan mitigar estas limitaciones y es parte del objetivo de este trabajo utilizarlas para crear una aplicación robusta.

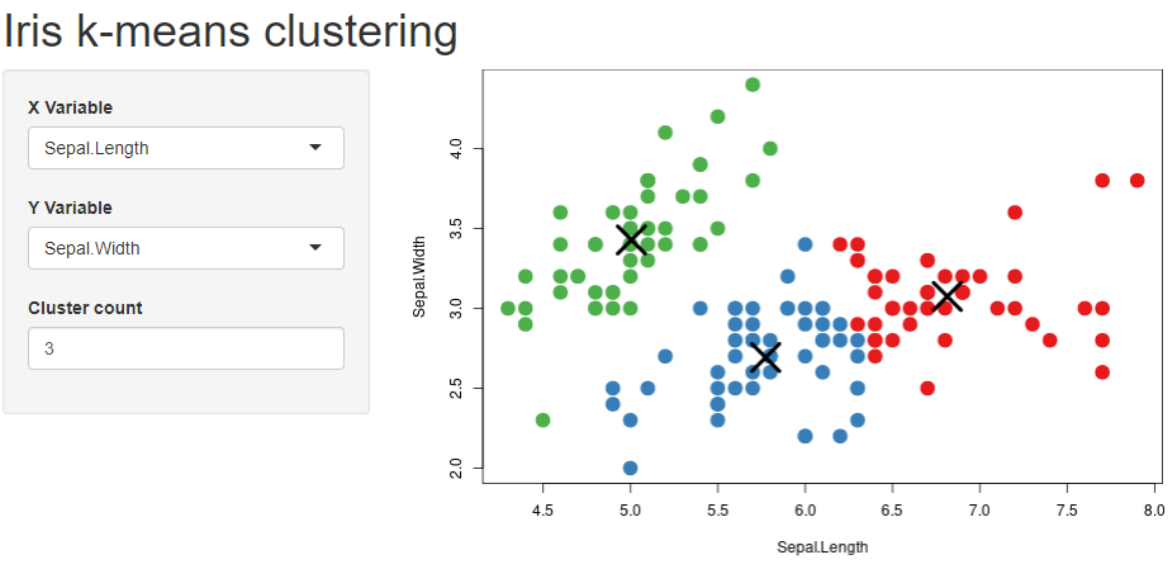


ILUSTRACIÓN 1 EJEMPLO DE UNA APLICACIÓN SHINY PARA LA VISUALIZACIÓN DE UN ALGORITMO DE CLÚSTERING

3.1.1. REACTIVIDAD

La interactividad en *Shiny* se obtiene a través del paradigma de la programación reactiva. La reactividad es un concepto central en *Shiny* y es el modo en que el paquete nos proporciona métodos de interacción con el usuario. Una variable reactiva (o, en sus términos de *Shiny* un *Reactive Producer*) en *Shiny* es un objeto similar al sujeto en el patrón observador, que notifica a sus observadores (o *Reactive Consumers*) cuando se produce un cambio sobre él. Un ejemplo gráfico del flujo reactivo se encuentra en la Ilustración 2.

En *Shiny* todos los inputs de usuario son *Reactive Producers*. *Shiny* nos proporciona dos tipos de Consumidores para manejar los datos reactivos:

- *Reactive*: La función reactive encapsula una expresión (reactiva o no) y devuelven un objeto reactivo. Al usarlas como consumidor sirven para realizar cálculos intermedios entre un objeto reactivo y su resultado final. Lo interesante de ellos es que su evaluación es perezosa, es decir, solo se evalúan si otro consumidor de reactividad requiere su valor.
- *Observe*: La función observe encapsula una expresión en un objeto observador. Este objeto reejecuta la expresión pasada como parámetro cada vez que alguna de sus dependencias reactivas notifica un cambio. Estos objetos observadores tienen dos diferencias fundamentales sobre los reactivos: no pueden devolver ningún valor y su evaluación no es perezosa, por lo que resultan útiles para modificar la interfaz de usuario.

En *Shiny*, además, todos los elementos de entrada del objeto interfaz están vinculados a Reactive Produces y todos los elementos de salida se asocian a *Reactive Consumers*, por lo que las operaciones básicas de E/S se limitan a grandes rasgos a una operación de lectura o asignación de una variable.

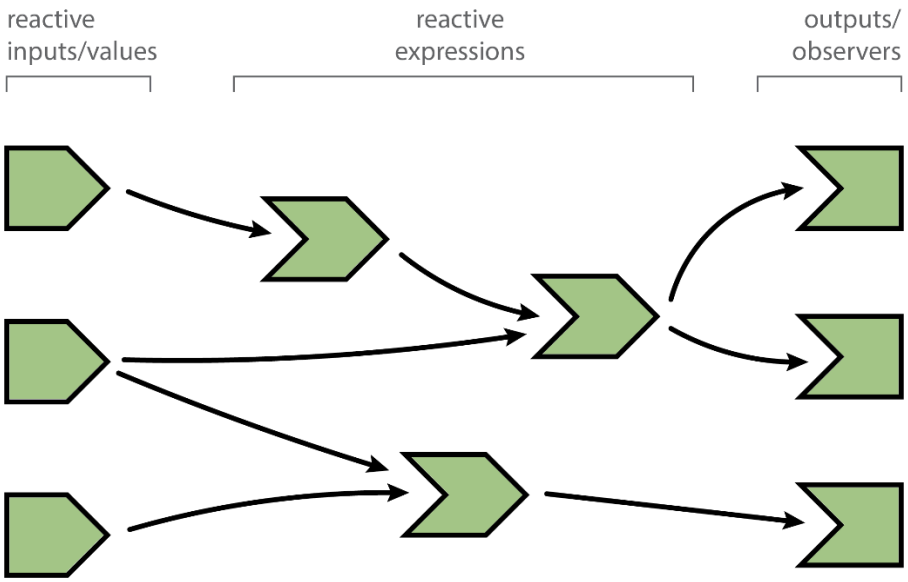


ILUSTRACIÓN 2 REPRESENTACIÓN GRÁFICA DE UN FLUJO DE REACTIVIDAD [4]

4. EL REAL DECRETO 1801/2008

El Real Decreto 1801/2008 es la norma actualmente vigente en España que establece las normas necesarias para garantizar que los consumidores, al adquirir productos envasados, tengan la garantía de que la cantidad indicada en el envase (i.e. cantidad nominal) muestra fielmente el contenido efectivo de éste.

Este Real Decreto, surge de la evolución de las normas anteriores RD 1472/1989 y Real Decreto 723/1988 [5] relativo al control del contenido efectivo de los productos alimenticios motivado por la necesidad de trasponer las Directiva 2007/45/CE [6] de la Unión Europea que obliga a fabricantes y distribuidores a realizar un control de calidad mínimo sobre sus procesos de envasado.

Para facilitar a los consumidores la identificación de los productos envasados siguiendo los procesos de control de contenido efectivo contenidos en este Real Decreto, se establece una marca común a toda la Unión Europea que debe encontrarse en los envases (Ilustración 3).

Además, en este Real Decreto se derogan las gamas de cantidades nominales que regían anteriormente (excepto en algunas bebidas alcohólicas), se establece un régimen sancionador y se fija cómo debe indicarse al consumidor que la cantidad nominal indicada en el envase cumple con la normativa. A efectos de este trabajo, nos centraremos en la norma que regula el proceso de calidad.

ILUSTRACIÓN 3 MARCA QUE INDICA QUE SOBRE ESTE PRODUCTO HAN SIDO APLICADOS LOS MÉTODOS DEL RD1801/2008

4.1. APLICACIÓN

Según el artículo segundo del Real Decreto, estos preceptos son de obligado cumplimiento para todos los artículos envasados orientados al consumidor final y destinados a su venta en cantidades nominales unitarias constantes siempre que cumplan los siguientes requisitos:

a) Iguales a valores prefijados por el envasador.

b) Expresados en unidades de masa o volumen.

c) Iguales o superiores a cinco gramos o cinco mililitros e inferiores o iguales a 10 kilogramos o 10 litros.

Además, excepcionalmente, deja fuera de esta norma ciertos productos destinados a consumirse fuera de la Unión Europea.

4.2. PRINCIPIOS GENERALES

Como idea general, en su artículo séptimo, el RD establece que un lote de cualquier producto envasado debe cumplir lo siguiente:

a) Que la media del contenido efectivo de los envases no sea inferior a la cantidad nominal.

b) Que la proporción de envases con un error por defecto superior al máximo tolerado sea lo suficientemente pequeña para que permita a los lotes satisfacer los controles estadísticos de este real decreto.

c) Que ningún envase tenga un error por defecto superior al doble del error máximo por defecto tolerado.

4.3. ANÁLISIS DE NO CONFORMIDADES

El primer control consiste en realizar mediciones sobre una muestra aleatoria del lote (el tamaño de esa muestra puede consultarse en la Tabla 3) y comparar el resultado de la medición (contenido efectivo) contra unos valores límite, obtenidos de la Tabla 1. Este análisis controla esencialmente que la variabilidad de la muestra no resulte en una pérdida de producto significativa para el consumidor.

El error máximo posible corresponde al doble del error máximo tolerado indicado en la Tabla 1. Si se encontrara algún envase con un contenido efectivo menor que este umbral, el lote se rechaza automáticamente.

Pero si se encontraran defectos, es decir, envases con un contenido efectivo menor que este umbral, pero mayor que el error máximo tolerado, se debe comparar el número de defectos encontrados contra los criterios de aceptación y rechazo de la Tabla 3.

Si el número de no conformidades fuera menor o igual que el criterio de aceptación el lote se acepta y si fuese mayor o igual que el criterio de rechazo el lote debe rechazarse. En el caso en el que un lote no cumpla ninguno de estos requisitos debe tomarse una nueva muestra y realizar con ella un análisis análogo.

En la Ilustración 4 puede verse un resumen de los criterios de aceptación y rechazo de un envase.

Cantidad nominal en gramos o en mililitros	Errores máximos por defecto tolerados			
	Masa		Volumen	
	Porcentaje – Cantidad nominal	En gramos	Porcentaje – Cantidad nominal	En mililitros
De 5 a 50	9,0	-	9,0	-
De 51 a 100	-	4,5	-	4,5
De 101 a 200	4,5	-	4,5	-
De 201 a 300	-	9,0	-	9,0
De 301 a 500	3,0	-	3,0	-
De 501 a 1.000	-	15,0	-	15,0
De 1.001 a 10.000	1,5	-	1,5	-

TABLA 1 ERRORES MÁXIMOS TOLERADOS SEGÚN EL REAL DECRETO

Tamaño del lote	Muestras		
	Orden	Tamaño	Tamaño acumulado
De 100 a 500	1.º	30	30
	2.º	30	60
De 501 a 3.200	1.º	50	50
	2.º	50	100
De 3.201 o más	1.º	80	80
	2.º	80	160

TABLA 2 TAMAÑOS DE MUESTRA

Tamaño del lote	Número de envases deficientes		
	Orden	Criterio de aceptación	Criterio de rechazo
De 100 a 500	1.º	1	3
	2.º	4	5
De 501 a 3.200	1.º	2	5
	2.º	6	7
De 3.201 o más	1.º	3	7
	2.º	8	9

TABLA 3 CRITERIOS DE ACEPTACIÓN Y RECHAZO

4.4. ANÁLISIS DE LA MEDIA

El segundo control consiste esencialmente en un intervalo de confianza sobre la media del lote utilizando los valores de la muestra. Si el cómputo del intervalo concluye que la media del lote iguala o supera la cantidad nominal con una probabilidad del 95%, el lote se acepta, en caso contrario, debe ser rechazado.

El real decreto nos proporciona una fórmula para la estimación de la desviación estándar y el valor crítico de la distribución de *t-student* que se debe usar para el cálculo según el tamaño muestral.

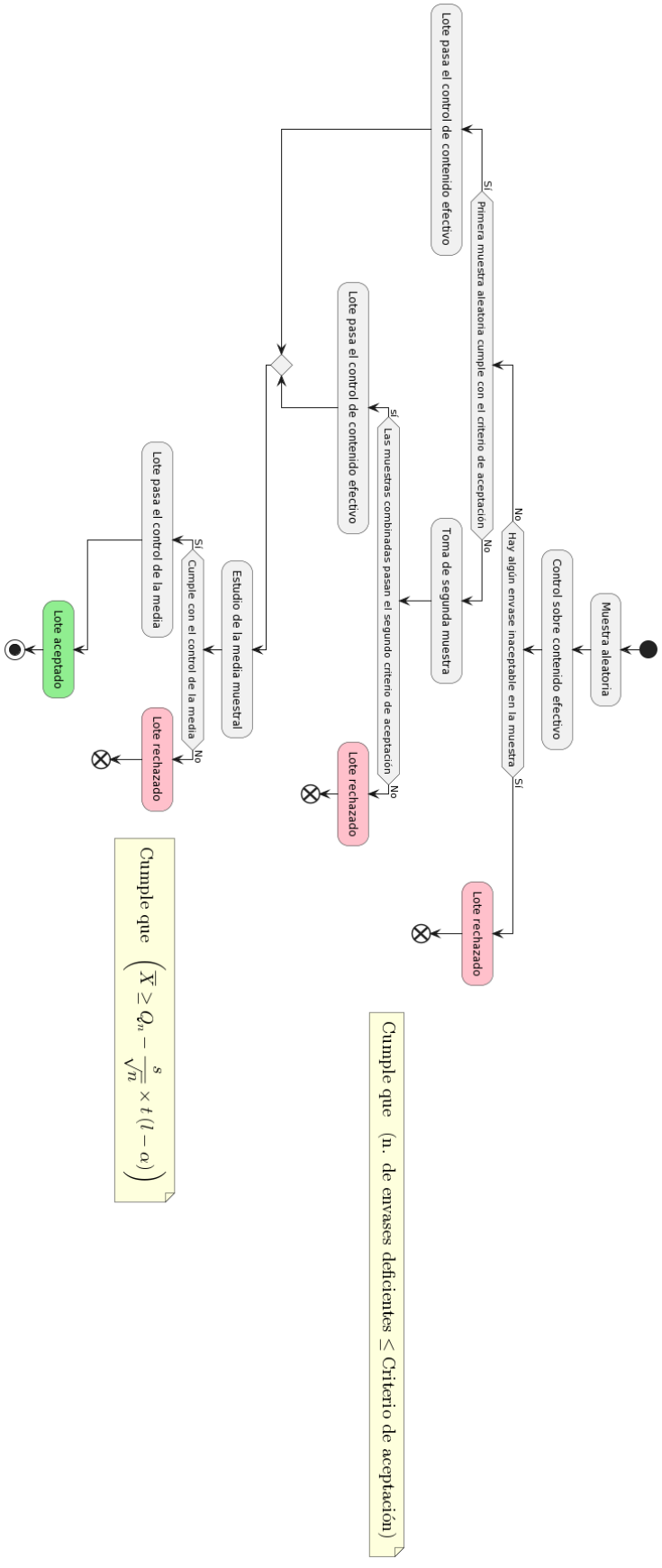


ILUSTRACIÓN 4 RESUMEN DE LOS CRITERIOS DE ACEPTACIÓN Y RECHAZO PARA UN LOTE SEGÚN EL REAL DECRETO

5. OBJETIVOS

El objetivo principal de este trabajo es realizar una aplicación en usando *R* y *Shiny* que implemente los análisis recogidos en el RD 1801/2008 sobre el contenido efectivo de productos envasados. Este programa estará pensado para que un usuario no técnico pueda realizar los análisis legalmente requeridos.

Como objetivos secundarios pretendemos analizar la viabilidad, restricciones y problemática de *R* y *Shiny* a la hora de implementar este tipo de aplicaciones.

Además de esto pretendemos que este trabajo sirva como guía metodológica para que profesionales fuera del ámbito de la ingeniería del software, pero con conocimientos de R puedan realizar sus propias aplicaciones *Shiny*. Por ello, durante el desarrollo procuraremos usar exclusivamente el lenguaje *R*, evitando insertar en la aplicación código *CSS*, *HTML* o *Javascript*.

6. ANÁLISIS DE ALTERNATIVAS

Antes de comenzar el desarrollo, hemos intentado estudiar qué otras aplicaciones existen en el mercado a disposición de las empresas que necesitan, en línea con lo dispuesto con el Real Decreto, realizar el control de calidad requerido para comercializar productos envasados.

Una búsqueda intensiva no arroja resultados sobre aplicaciones independientes que auxilien a los técnicos de control de calidad en estas tareas, sin embargo, si encontramos diferentes alternativas en forma de funcionalidades incluidas en sistemas integrados como ERPs o maquinaria industrial conectada.

6.1. MÓDULOS EMBEBIDOS EN SOLUCIONES INTEGRALES

Utilidades dentro de las distintas soluciones ERP y programas embebidos dentro de instrumentación industrial: En este apartado encontramos distintas alternativas para este problema, no como programas independientes sino como módulos embebidos tanto en soluciones de datos integrales como en maquinaria especializada.

La ventaja competitiva de estos módulos es que, si la empresa ya es usuaria del sistema en el que se acopla, se integra muy fácilmente en el flujo de trabajo de la organización, pudiendo generar flujos de trabajo totalmente automatizados gracias a sistemas de medición automáticos que se encuentran fácilmente en el mercado.

La desventaja principal proviene del coste. Una solución de este tipo sea únicamente mediante software o incluyendo maquinaria industrial conectada en red supone una inversión significativa. No podemos, sin embargo, proporcionar datos específicos a este respecto dado que esto depende en gran medida de contratos confidenciales entre empresas, sin embargo, *softwarepath* estima el coste medio en unos 9.000$ por usuario y año [7] aunque los costes indirectos de la adopción de sistemas de este estilo pueden ser aún mayores. [8]

- *Bizerba* (*BRAIN2*): Ofrece un software general de control de procesos modular. Entre las funcionalidades ofrece la gestión del Real Decreto en cuestión. [9]
- *Dival*: Ofrece una solución software integrada en los equipos de pesaje que suponen el negocio principal de la compañía. [10]
- *ASM*: Es una consultora tecnológica que ofrece un software integral de gestión de producción. Entre las funcionalidades que ofrece es un módulo que realiza los cálculos necesarios para cumplir con los requisitos legales del Real Decreto. [11]
- *Giropes* (*GISCALE*): Empresa dedicada a la comercialización de equipos de pesaje industrial. Ofrecen un software que conecta sus equipos y proporciona monitorización constante sobre los estándares de calidad de la producción. [12]
- *DAII S.L (Open Weight*, Ilustración 5): Es otra empresa cuya actividad principal es la comercialización de equipos. Ofrecen un software distribuido llamado *Open Weigh* pensado para que diferentes terminales de pesaje operados independientemente envíen las mediciones a terminales de gestión dónde se monitorizan los parámetros deseados. [13]

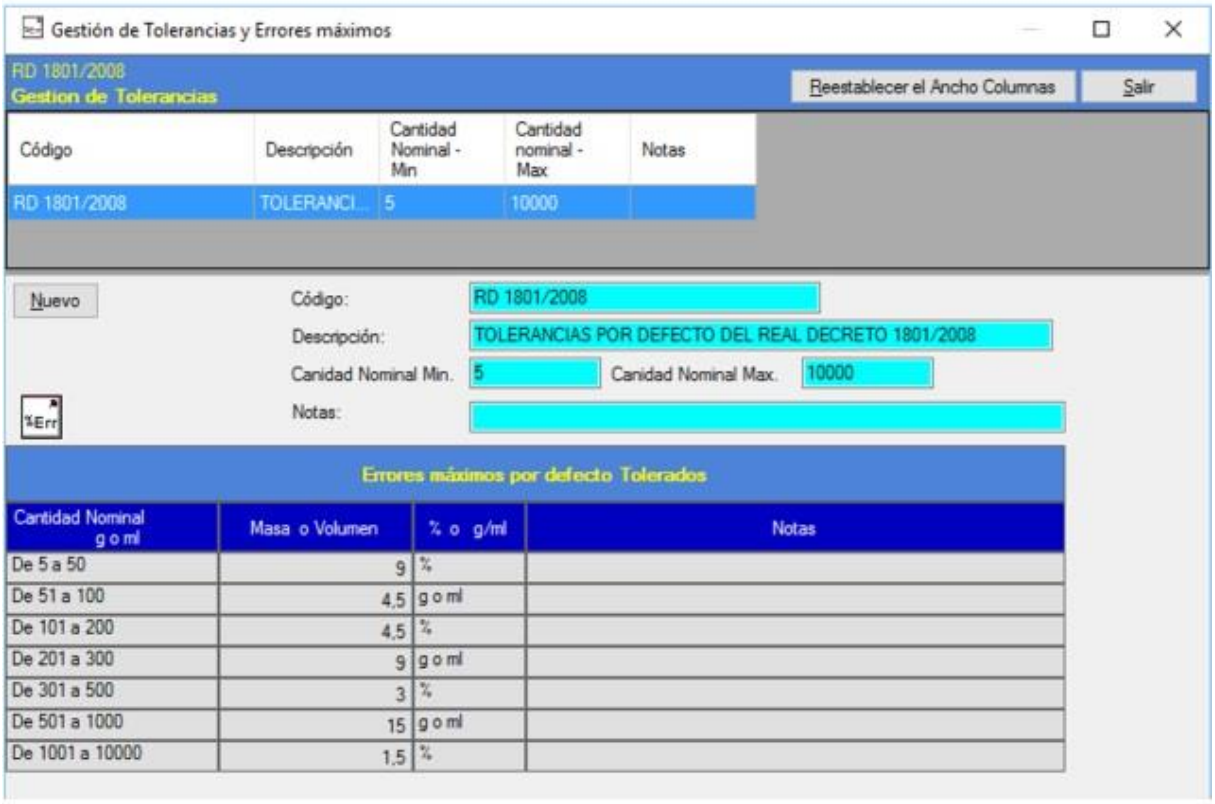


ILUSTRACIÓN 5 CAPTURA DEL PROGRAMA OPEN WEIGHT

En general estas soluciones ofrecen a sus clientes:

a) Extracción de los datos de las muestras directamente desde la línea de producción

b) Monitoreo en tiempo real de la producción.
c) Análisis de no-conformidades con:
    a. Visualización de los elementos de la muestra y si cumplen o no con los criterios.
    b. Visualización gráfica de la muestra.
    c. Extracción de un informe de cumplimiento.
d) Análisis de la media.
e) Visualización de otros estadísticos del lote.
f) Almacenamiento integrado del histórico de análisis.

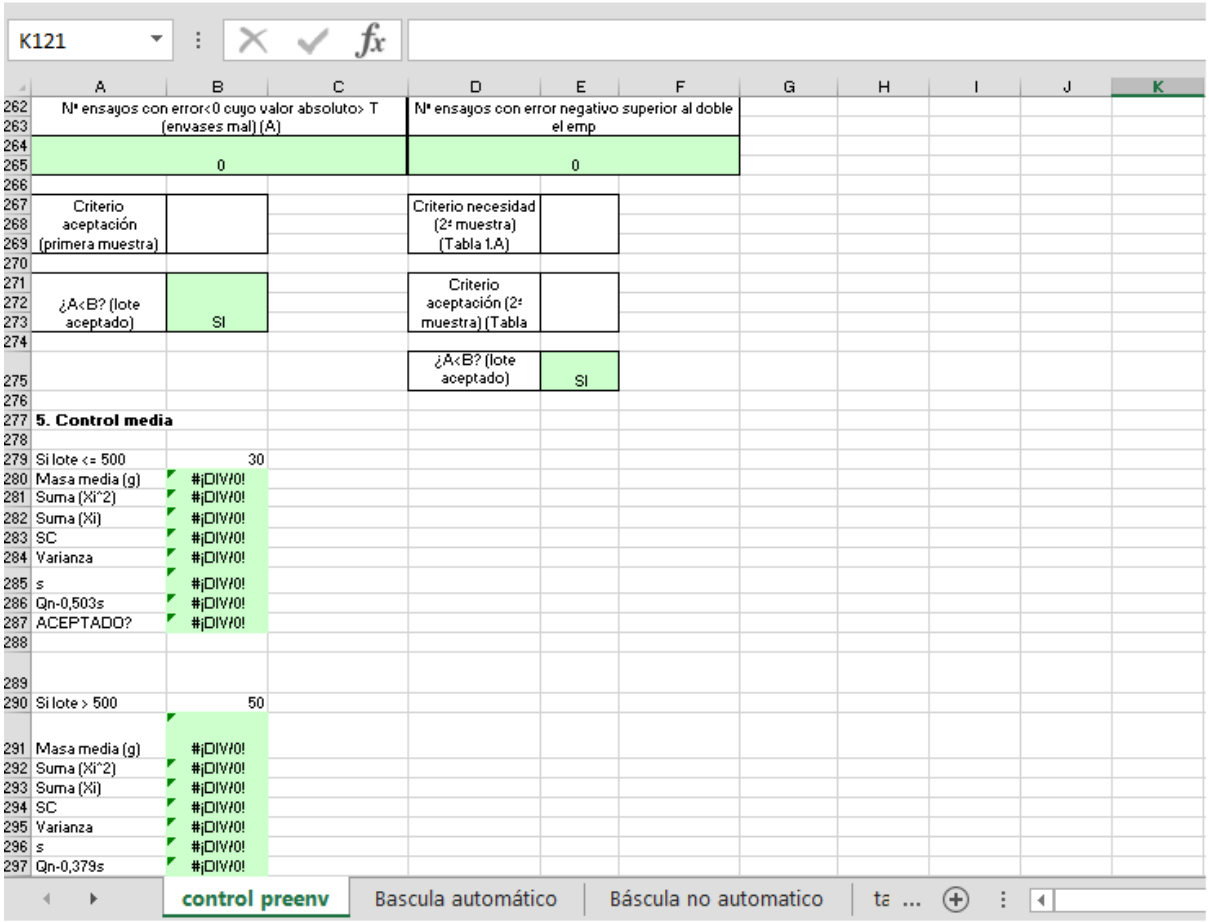
6.2. SOLUCIONES AD-HOC SOBRE SOFTWARE NO ESPECÍFICO

Por otro lado, y dado el coste de las soluciones anteriores, es muy probable que muchas empresas opten por utilizar soluciones implementadas sobre software estadístico genérico como SPSS, Minitab o incluso Excel como hizo la autoridad inspectora de la Comunidad de Madrid durante 2018 [14] (Ilustración 6). Sin embargo, dada la naturaleza de éstas resulta difícil encontrar ejemplos públicos en los que inspirar nuestro desarrollo.

La ventaja de estas aplicaciones es principalmente su coste y que, para empresas pequeñas y medianas ofrecen una gran cantidad de personalización. Esto favorece que la experiencia de usuario se amolde lo máximo posible al flujo de trabajo que los usuarios acostumbran.

Además, probablemente los usuarios de estas soluciones ya utilizaban estos programas con anterioridad, por lo que los costes indirectos en formación son casi nulos.

Como desventajas se puede mencionar la limitada automatización que pueden ofrecer estas soluciones, pudiendo ser propensas a errores humanos si no se diseñan con cautela y el mayor gasto en horas de trabajo a largo plazo y suele ser necesario mantener en plantilla personal capaz de mantenerlas. La naturaleza de éstas también puede provocar una dependencia excesiva de software o versiones de software propietario sobre la que las empresas usuarias tienen poco control, especialmente según aumentan las funcionalidades de estas aplicaciones y con ello, su complejidad.



	A	B	C	D	E	F	G	H	I	J	K
262	Nº ensayos con error<0 cuyo valor absoluto> T (envases mal) (A)			Nº ensayos con error negativo superior al doble el emp							
263											
264											
265	0			0							
266											
267	Criterio aceptación (primera muestra)			Criterio necesidad (2ª muestra) (Tabla 1.A)							
268											
269											
270											
271	¿A<B? (lote aceptado)			Criterio aceptación (2ª muestra) (Tabla							
272		SI									
273											
274											
275				¿A<B? (lote aceptado)	SI						
276											
277	**5. Control media**										
278											
279	Si lote <= 500	30									
280	Masa media (g)	#¡DIV/0!									
281	Suma (Xi^2)	#¡DIV/0!									
282	Suma (Xi)	#¡DIV/0!									
283	SC	#¡DIV/0!									
284	Varianza	#¡DIV/0!									
285	s	#¡DIV/0!									
286	Qn-0,503s	#¡DIV/0!									
287	ACEPTADO?	#¡DIV/0!									
288											
289											
290	Si lote > 500	50									
291	Masa media (g)	#¡DIV/0!									
292	Suma (Xi^2)	#¡DIV/0!									
293	Suma (Xi)	#¡DIV/0!									
294	SC	#¡DIV/0!									
295	Varianza	#¡DIV/0!									
296	s	#¡DIV/0!									
297	Qn-0,379s	#¡DIV/0!									

ILUSTRACIÓN 6 HOJA EXCEL UTILIZADA POR LA COMUNIDAD DE MADRID PARA LA INSPECCIÓN.

6.3. SOLUCIÓN PROPUESTA

Analizando estos precedentes creemos que la publicación de una aplicación R que resuelva esta problemática es adecuada por los siguientes motivos:

1) Diferenciación: Nuestra aplicación, aunque ofrezca una funcionalidad básica similar es una solución basada en web, lo que la hace apta para cualquier plataforma que admita un navegador.
2) Conveniencia: Nuestra aplicación implementa los procesos recogidos por el Real Decreto sin necesidad de adaptación o mantenimiento por parte de las empresas usuarias.
3) Versatilidad: El paquete desarrollado incluirá la implementación de los procesos del Real Decreto como funciones independientes permitiendo a las empresas utilizarlo como dependencia para el desarrollo de soluciones específicas.
4) Coste: Cualquiera de las soluciones arriba descritas implica un coste humano o monetario significativo. Las pequeñas empresas de producción limitada, especialmente del sector primario pueden beneficiarse de nuestra aplicación a coste cero.

7. METODOLOGÍA

Para este proyecto se hemos decidido seguir una metodología ágil basada en Scrum. Esta adaptación la hemos realizado teniendo en cuenta las especiales características especiales del proyecto, esto es, que el equipo de desarrollo es de una sola persona y hay un horizonte temporal claro para la entrega del producto.

La inspiración de este flujo de trabajo ha sido, por supuesto, la metodología Scrum, muy utilizada en la industria con el manifiesto ágil como base [15].

Antes de comenzar el desarrollo propiamente dicho, empezamos definiendo el alcance del proyecto, para tener un punto de partida a la hora de decir nuestras historias de usuario. Procuramos hacerlo en una lista con frases cortas para que el equipo de desarrollo las interiorizara rápidamente. Estas son:

1) Se desea realizar una aplicación en R usando la librería *Shiny*.
2) La aplicación debe permitir a un usuario realizar las operaciones necesarias para cumplir con el Real Decreto 1801/2008.
3) El desarrollo debe realizarse siguiendo buenas prácticas de la ingeniería del Software.

Basándonos en esto, el primer paso fue establecer una metodología de trabajo adecuada. Nos plantemos las siguientes alternativas:

1) Scrum: Aunque es quizás la metodología más popular dentro de la industria del software, consideramos que las ceremonias no tendrían sentido para un equipo unipersonal.
2) Extreme Programming: Aunque nos resultaba atractiva, el enfoque centrado en feedback continuo del cliente (entendido como el Tutor) y del equipo no encajaba con las características del proyecto.

Finalmente decidimos adoptar una metodología Ágil propia basada en *Scrum,* pero eliminando gran parte de las ceremonias. Decidimos dividir la funcionalidad en historias de usuario e ir implementándolas en *Sprints* sucesivos de dos semanas, sin embargo, descartamos completamente los roles marcados por la metodología Scrum y reducimos las ceremonias a la mínima expresión:

1) *Daily meetings*: Descartadas por completo.
2) *Sprint retrospective*: Al final de cada sprint se reservaría un pequeño tiempo al final del sprint para reflexionar sobre la evolución del proyecto e implementar los cambios que se consideren en la metodología.
3) *Sprint planning*: Al principio de cada sprint se asignarían una serie de historias de usuario al sprint en función de su prioridad.

Además de esto e inspirándonos en los principios del Manifiesto Ágil, decidimos que el proyecto siguiera los siguientes principios, de forma similar a lo propuesto por *EXTreme Programming*:

1) Mejora continua.

2) Entrega continua de valor.
3) Testing extensivo.

Sin embargo, decidimos posponer la concreción de estos principios a la definición de una serie de Historias Técnicas más adelante.

7.1. REQUISITOS NO FUNCIONALES

Con el alcance y la metodología básica definida, comenzamos a definir definimos una serie de requisitos no funcionales y principios de diseño. Algunas de estas restricciones se trasformarán más adelante en historias técnicas que deban ser implementadas y otras deberán integrarse en el flujo de trabajo de forma transversal. Estos requisitos y su motivación son los siguientes:

1) El proyecto debe ser desarrollado en *Shiny*: Motivado por el alcance del proyecto.
2) Debe limitarse lo máximo posible el código en otros lenguajes para desarrollar la aplicación: Dado que el objetivo del trabajo es fundamentalmente demostrar las capacidades de la librería *Shiny* y ésta encapsula totalmente el código *HTML* y *JS* generado no nos pareció adecuado usar *Shiny* como excusa para desarrollar una aplicación web estándar.
3) La aplicación generada debe ser un paquete de R funcional: Aunque esto no es imprescindible para generar una aplicación *Shiny*, parte del objetivo es seguir las mejores prácticas posibles de la Ingeniería del Software, y el modo más estándar de compartir funcionalidad de R es mediante paquetes en *CRAN*.
4) La aplicación debe ser testada automáticamente: Motivado por los principios establecidos en la metodología.
5) El *lead time* (esto es, el tiempo de entrega de valor) del desarrollo debe ser el menor posible: Motivado por los principios de la metodología.
6) Un usuario familiarizado con el Real Decreto 1801/2008 debe poder utilizar la funcionalidad básica sin consultar ningún manual, es decir, la aplicación debe ser intuitiva.
7) Un usuario familiarizado con el control estadístico de calidad debe poder comprender el flujo básico del RD1801/2008 sin consultarlo.
8) Un usuario familiarizado con la herramienta debe poder realizar el proceso de control en el mínimo número de clicks posibles.
9) El tiempo de respuesta de la aplicación debe ser razonable.
10) La aplicación no debe almacenar ninguna clase de dato. Esto es debido a que planteamos que los datos de producción de una empresa suelen ser de carácter confidencial y su almacenamiento podría provocar reticencias entre los usuarios potenciales y obligar al equipo de desarrollo a realizar consideraciones de seguridad fuera del alcance de este trabajo.

Las historias técnicas que se generaron a partir de estos requisitos fueron las siguientes:

- Como dev quiero tener test de ejemplo en los que basarme
- Como dev quiero que la app ejecute todos los test antes de mergear a main.

- Como dev quiero que la aplicación se despliegue automáticamente en producción.
- Como dev quiero que la aplicación se despliegue automáticamente en un servidor de pruebas.
- Como dev quiero tener facilidades para hacer una app accesible.

7.2. REQUISITOS FUNCIONALES

Los requisitos funcionales son los componentes esenciales de una aplicación. Definen las interacciones específicas que puede tener un usuario con ella. En nuestro caso, decidimos definir estos requisitos usando Historias de Usuario que se centran en el valor aportado al usuario de los requisitos.

Para ello, el primer paso fue definir las características de los usuarios de nuestra aplicación. Por las características de nuestra aplicación y la decisión, tomada previamente de que no íbamos a almacenar ninguna clase de información, solo íbamos a tener un tipo de usuario, que podría realizar todas las operaciones que ofreciéramos.

Este usuario, naturalmente, debe estar familiarizado, al menos, con el análisis del contenido efectivo de envases a nivel general, con los procedimientos de muestreo y con la terminología empleada en el Real Decreto 1801/2008. Sin embargo, no asumimos que conoce los procedimientos exactos del Real Decreto.

Una vez definido esto, escribimos una serie inicial de historias de usuario (Tabla 4). No nos centramos en lo realizable en este trabajo, dado que el foco principal de la metodología está en maximizar el valor aportado mediante la priorización, no la definición a priori de una hoja de ruta.

Historias de usuario iniciales
Como usuario quiero ver una hoja resumen de mi análisis para poder valorar el estado del proceso de un vistazo.
Como usuario quiero poder ver si mi plan de muestreo es equivalente al oficial
Como usuario quiero poder usar un plan de muestreo personalizado
Como usuario quiero poder elegir entre dos modos visuales para poder distinguir bien los elementos de la web.
Como usuario quiero ver una guía de uso paso a paso para aprender a utilizarla.
Como usuario quiero que la app me indique el riesgo de no pasar el proceso de calidad para poder tomar decisiones sobre mi proceso de producción.
Como usuario potencial quiero ver un ejemplo de toda la funcionalidad para decidir si quiero usarla
Como usuario quiero que mi análisis pueda recrearse en función de una url para compartirlo con mis compañeros

Como usuario quiero que la app me permita descargar un informe de todo el análisis para poder almacenarlo en mi histórico.
Como usuario quiero que la app me muestre la tabla de la muestra marcando las no conformidades para poder diferenciarlas.
Como usuario quiero que la app muestre los resultados del análisis de la media con un gráfico para facilitar la interpretación de este.
Como usuario quiero que la aplicación muestre los resultados del análisis de nC con un gráfico para facilitar la interpretación de este.
Como usuario quiero poder introducir los datos de mi segunda muestra para el análisis de no conformidades para rehacer el análisis.
Como usuario quiero poder introducir los datos de mi muestra inicial para el análisis de no conformidades para realizar el análisis.
Como usuario quiero introducir los datos de mi lote y que la app me guíe en los siguientes pasos para no cometer errores.

TABLA 4 HISTORIAS DE USUARIO DEFINIDAS AL COMIENZO DEL DESARROLLO

7.3. DISEÑO DE LA INTERFAZ

Antes de comenzar a codificar decidimos crear un prototipo puramente visual de la aplicación. Para ello decidimos utilizar PowerPoint (Ilustración 7). A pesar de ser una herramienta para crear presentaciones, ofrece una interfaz intuitiva y que permite la creación de dibujos mediante formas predefinidas con gran capacidad de personalización.

Además de PowerPoint, existen aplicaciones específicas para realizar esta tarea. Sin embargo, muchas de ellas tienen una complejidad mayor de la que nos parecía necesaria para esta aplicación, además de ser, en general, de pago. Algunas de estas aplicaciones son:

- Balsamiq
- Sketch
- Adobe XD

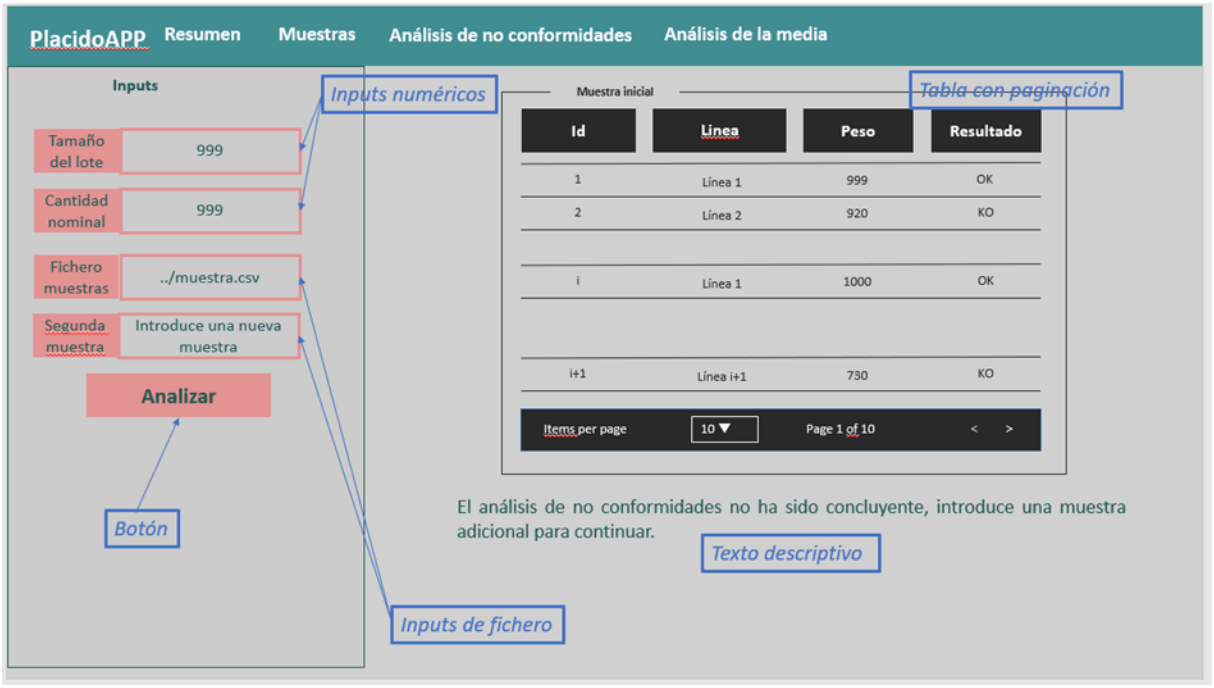


ILUSTRACIÓN 7 ESQUEMA DE LA INTERFAZ DISEÑADO USANDO POWER POINT.

El diseño visual de nuestra aplicación se encontraba limitado por las opciones proporcionadas por *Shiny*. Sin embargo, para proporcionar una experiencia adecuada a nuestros usuarios nos pareció importante definir tres principios de diseño:

1) Que un usuario avanzado pudiera realizar el análisis con el menor número de clicks posible.
2) Que todo gráfico o tabla se acompañara de una explicación de texto para facilitar la accesibilidad.
3) Que el flujo de uso de las pestañas fuera lineal de izquierda a derecha.

En cuanto a accesibilidad, procuramos seguir las directivas de accesibilidad proporcionadas por el W3C [16] como norma general. Sin embargo, relegamos el cumplimiento completo de estas recomendaciones a una historia técnica, dado que el desarrollo de algunas de ellas no es trivial usando *Shiny*.

7.4. GESTION DE TAREAS

Para la gestión de tareas de este proyecto hemos decidido utilizar la plataforma Jira dado que es una de las herramientas de gestión de proyectos más populares [17] y ofrece una enorme versatilidad. Esto nos permitió adaptar la herramienta de gestión al flujo de trabajo que consideramos más adecuado para nuestro proyecto y no al revés. Dada la implementación ad hoc de la metodología Scrum que utilizamos en este proyecto, esta flexibilidad era esencial para que la herramienta fuera realmente útil. Además, con la posibilidad de conectar el proyecto Jira con nuestro repositorio en GitHub conseguimos que la gestión del proyecto fuera más cohesiva.

Jira es una aplicación de la australiana Atlassian, fundada en el año 2002 por Scott Farquhar y Mike Cannon [18]. Atlassian es conocida en la industria del software por sus productos de colaboración y productividad en los equipos de trabajo. Jira, fue diseñada como una herramienta de seguimiento de bugs e incidencias, aunque ha evolucionado hasta convertirse en una plataforma integral de gestión de proyectos y ciclo de vida del software.

Para la configuración del proyecto de Jira nos hemos basado en las tarjetas tipo Historia de Usuario, en las que escribimos todas las historias que habíamos diseñado al comienzo del proyecto.

7.4.1. TARJETAS DE HISTORIA

Al comienzo del proyecto configuramos las tarjetas de historia de forma que sólo pudieran verse y modificarse los atributos que nos interesaban para la gestión del proyecto, esto es:

1) Título.
2) Responsable.
3) Autor.
4) Prioridad (en formato *MoSCoW*)
5) Tiempo empleado.
6) Rama de implementación.
7) Subtareas.

Los atributos de responsable y autor los eliminamos en el primer *sprint retrospective* dado que no aportaban ninguna información en este caso, quedando pendiente su inclusión más adelante si fuera necesario.

En el caso de las subtareas, durante los primeros Sprints se estableció un flujo de trabajo por el cual se creaban al menos tres tarjetas por cada Historia de Usuario que pasara a "En Progreso": análisis, implementación y revisión dejando la posibilidad abierta de crear más tareas de implementación si se detectaban una necesidad de suficiente entidad. Al final del tercer sprint decidimos eliminar esta práctica porque añadía burocracia que no siempre se llevaba con el rigor necesario, por tanto, ni aportaba información ni facilitaba el desarrollo.

7.4.2. FLUJOS DE TRABAJO

Con la ayuda de la herramienta de flujos de trabajo de Jira (Ilustración 8) definimos los estados en los que podía encontrarse cada historia de usuario en cuatro:

1) Por hacer: Historias de usuario en el backlog del proyecto o del sprint sobre las que aún no se ha comenzado a trabajar.
2) En progreso: Historias de usuario en las que se está trabajando en este momento y tienen una rama activa en el repositorio del proyecto.
3) En revisión: Historias de usuario completadas y con una *pull request* activa sobre su rama de desarrollo.
4) Bloqueado: Historias de usuario sobre las que no se puede trabajar a la espera de realizar alguna otra tarea o resolver algún problema.

5) Hecho: Historias de usuario completadas y cuya rama de desarrollo ha sido incorporada a la rama principal.

Además, se establecieron una serie de relaciones entre estos estados o acciones que conllevan un cambio de un estado a otro. Las relaciones que se crearon pueden consultarse en la Ilustración 8.

ILUSTRACIÓN 8 FLUJO DE TRABAJO DEFINIDO PARA LAS HISTORIAS DE USUARIO TAL Y COMO SE MUESTRA EN JIRA.

Al comienzo del proyecto también establecimos un flujo de trabajo similar para las subtareas descartando el estado "En revisión". Este flujo se mantuvo, aunque lo utilizamos muy puntualmente.

Por otro lado, Jira ofrece a sus usuarios la posibilidad de crear automatizaciones para sus proyectos mediante *Jira Automation* una herramienta *no code*. Para este proyecto, creamos las siguientes macros (Ilustración 9) basándonos en las opciones predefinidas que proporciona Atlassian:

1) Mover la historia a "En curso" cuando se cree su rama asociada.
2) Mover la historia a "Hecho" cuando se acepte su *pull request* asociada.
3) Crear las subtareas de una Historia de Usuario.
4) Cerrar las subtareas asociadas a una Historia de Usuario cuando esta pasa a "Hecho"

Al dejar de utilizar las subtareas por defecto de una historia de usuario se deshabilitó la tercera regla, aunque se mantuvo la cuarta dado que dejamos la puerta abierta a crear subtareas si la historia lo requería.

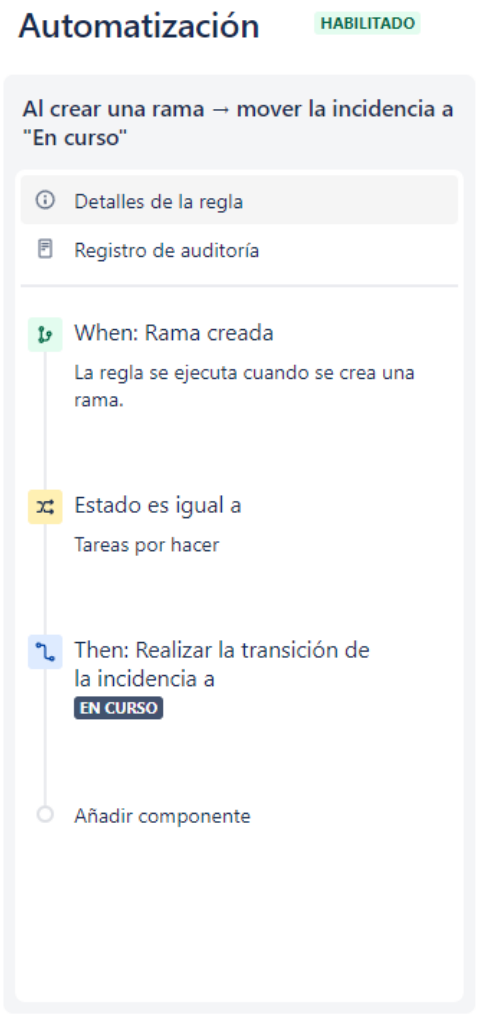


ILUSTRACIÓN 9 INTERFAZ DE EDICIÓN DE UNA AUTOMATIZACIÓN

7.4.3. TABLERO ÁGIL

Una vez decidido que la metodología de trabajo iba a implicar el uso de *Sprints* se configuró un tablero Ágil para el proyecto (Ilustración 10). Esto nos permitía definir diferentes *Sprints* y asignar, de forma iterativa las historias a realizar en cada sprint y el estado en el que se encontraban en cada momento. De esta forma, podíamos analizar de un solo vistazo el estado y el progreso del Sprint y cada tarea.

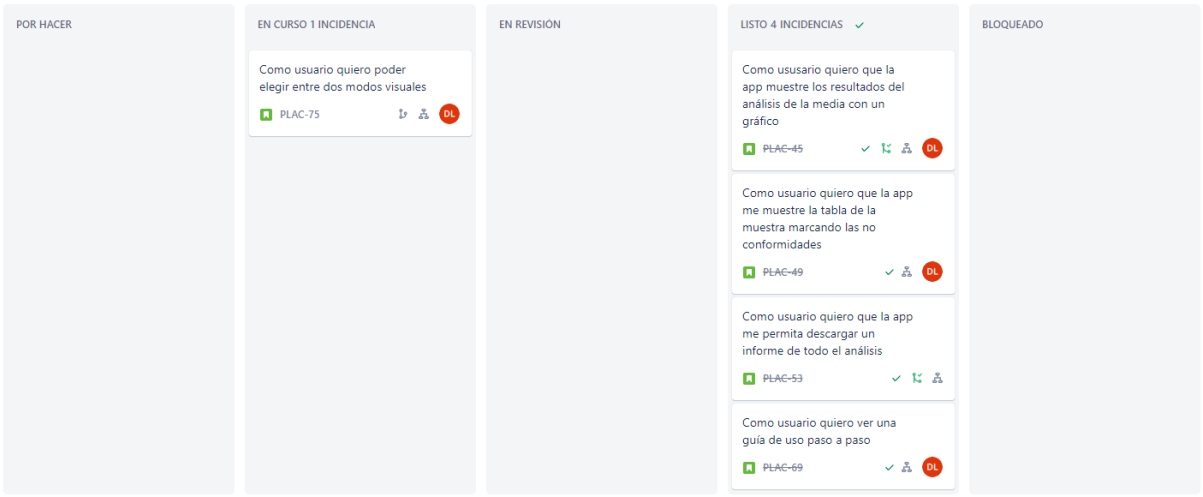


ILUSTRACIÓN 10 TABLERO ÁGIL CONFIGURADO EN JIRA

7.4.4. SINERGIAS DE JIRA

Atlassian ha añadido en Jira diferentes interconexiones con aplicaciones propias y con un ecosistema amplísimo de herramientas de terceros. Por ejemplo, Jira permite enlazar un proyecto con su propia herramienta de creación de bases de conocimientos (*Confluence*) o un repositorio en GitHub.

En nuestro caso, aunque valoramos la posibilidad de crear una pequeña wiki en *Confluence*, sí que usamos estas sinergias para enlazar con el repositorio del proyecto (Ilustración 11). Esto nos permitía crear, de forma automatizada, ramas de desarrolla basadas en historias de usuario y visualizar el estado de las *pull request* de estas desde Jira.

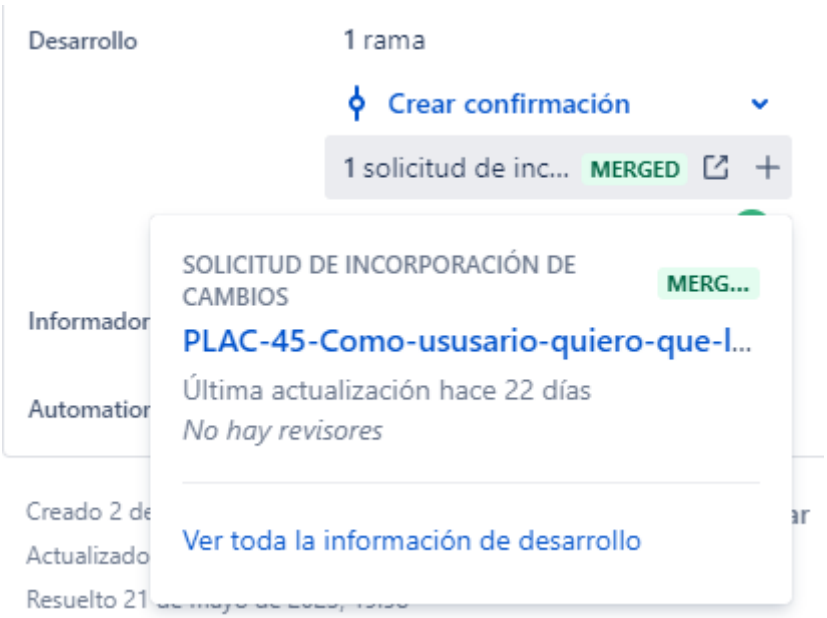


ILUSTRACIÓN 11 VISUALIZACIÓN DE UNA PULL REQUEST DESDE JIRA

7.5. CONTROL DE VERSIONES

Para el control de cambios sobre el código y como complemento a Jira para la versión de tareas hemos empleado *GitHub* dado que nos ofrece la capacidad de realizar un seguimiento de los cambios en el código a través de su funcionalidad de control de versiones (basada en *Git* [19]), gestionar las revisiones de código mediante su herramienta de *pull request* e implementar flujos automatizados de integración continua de una forma sencilla.

GitHub es una plataforma de desarrollo de software colaborativo fundada en 2008. Se diseñó para simplificar la colaboración entre muchos programadores en desarrollos de software de código abierto y es actualmente el repositorio de código más grande del mundo.

7.5.1. PULL REQUEST

Las *pull request* son una de las funcionalidades que forman el núcleo de *GitHub*. Permiten a los programadores solicitar la incorporación de código a la rama principal del repositorio de forma ordenada y motivada, por lo que son la base de la colaboración entre los distintos desarrolladores y la realización de revisiones de código.

En nuestro desarrollo hemos utilizado las *pull request* (Ilustración 12) como métodos para realizar revisiones de código antes de integrar una nueva funcionalidad en la herramienta. La idea ha sido crear una rama por cada historia de usuario que se deseara incluir y, una vez desarrollada realizar una *pull request* antes de incluirla en la rama principal.

ILUSTRACIÓN 12 EJEMPLO DE PULL REQUEST

7.6. INTEGRACIÓN Y ENTREGA CONTINUA

La Integración y Entrega Continuas (o *CI/CD*) es una práctica fundamental de la Ingeniería del Software. Se refieren a la integración dentro de los flujos de trabajo de los equipos de desarrollo la incorporación continua de los cambios realizados en el repositorio central y la entrega de dichos cambios al cliente.

Esto, entre otras cosas, disminuye la cantidad de trabajo necesario para integrar nuevas funcionalidades, aumenta la recurrencia con la que el cliente puede proporcionar *feedback* y facilita la detección y corrección de errores.

Para nuestro desarrollo, hemos diseñado un flujo de trabajo en los que en cada funcionalidad de ejecutan las pruebas automáticamente, y se sube la aplicación a un entorno de pruebas. Para ello hemos utilizado *GitHub Actions*.

7.6.1. GitHub Actions

GitHub Actions permite a los desarrolladores crear y ejecutar automatizaciones de tareas del ciclo de vida del software, como la compilación, la prueba y el despliegue de aplicaciones. Se puede configurar de forma que se activen en respuesta a eventos específicos en el repositorio, como la creación de una rama, o una *pull request*.

Los flujos de trabajo se definen en archivos *YAML* dentro del repositorio. Cada flujo de trabajo consta de una serie de instrucciones que se ejecutan secuencialmente sobre una variedad de entornos diferentes. Estos entornos se comportan esencialmente como instancias de máquinas virtuales, sobre las que se ejecutan scripts, instrucciones de Shell e incluso conjuntos de acciones creadas por la comunidad.

En nuestro caso, hemos utilizado *GitHub Actions* para establecer dos flujos de trabajo, uno que despliegue la aplicación en producción y otro que realice los chequeos oportunos y despliegue la aplicación en un entorno de pruebas.

8. EL DESARROLLO

8.1. DEFINICIÓN DE LA ARQUITECTURA

Aunque *Shiny* permite la creación de una aplicación de forma extremadamente rápida y no requiere de una estructura rígida en absoluto [20], parte del objetivo de este trabajo era utilizar las mejores prácticas de la ingeniería del software, teniendo en cuenta especialmente la mantenibilidad futura de la aplicación.

Por ello descartamos desde un inicio la creación de un código monolítico y decidimos utilizar la posibilidad de crear módulos que ofrecen las versiones más recientes de *Shiny*.

También contemplamos la posibilidad de desarrollar la aplicación como un paquete de estándar de R, pero esto nos obligaba primero, a realizar una serie de adaptaciones no triviales sobre el código y segundo, asignar recursos constantemente a cumplir los requisitos necesarios para ser un paquete de R de pleno derecho (es decir, cumplir con la lista de chequeos necesarios [21] (Tabla 5 Tabla resumen de los requisitos para que un paquete R esté bien formado); por lo que descartamos inicialmente esta posibilidad.

Más adelante, descubrimos el paquete *Golem*. *Golem* nos permitía beneficiarnos de ser un paquete de R bien formado facilitando una buena parte de la configuración inicial y el mantenimiento. El método de trabajo de *Golem* y el resto de los paquetes del *Golemverse* (un conjunto de librerías relacionadas bajo la misma filosofía), recogida en *Engineering Shiny* [22] se alineaba perfectamente con los objetivos de este trabajo. Esto nos animó definitivamente a realizar el desarrollo como un paquete de R.

Uno de los primeros inconvenientes que encontramos, es que un paquete R no puede contener ficheros fuente con caracteres no ASCII por motivos de portabilidad. Esto era un verdadero obstáculo, dado que la interfaz de usuario iba a contener texto en castellano y, en consecuencia, tildes de distinto tipo. Aunque estudiamos por un tiempo la posibilidad de usar un paquete de localización como *i18n* [23], esto nos forzaba a utilizar de base una localización en inglés que carecía de sentido para una aplicación destinada a los procesos marcados por una norma española. Finalmente nos decidimos a incluir una función de conversión en los ficheros de desarrollo incluidos en */dev*.

Estructura

El directorio del paquete existe.

El paquete está formado por ficheros fuente, no binarios.

No debe haber ficheros ejecutables en el paquete.

No hay ficheros ocultos en el paquete.

Los nombres de los ficheros son portables entre distintos sistemas operativos.

La operación de chequeo tiene permisos suficientes para trabajar con los ficheros del directorio.

Es posible instalar el paquete usando *R CMD INSTALL*.

El paquete instalado pesa menos de 5MB.

La estructura de directorios es correcta y no hay ficheros inesperados en el directorio raíz.

Subdirectorios
Todos los nombres de subdirectorios empiezan por minúscula excepto /R.
El contenido de inst/ no genera conflictos con el directorio raíz.
No hay ficheros inesperados dentro de los subdirectorios.
Metadatos
El fichero DESCRIPTION está bien formado
El fichero DESCRIPTION no contiene caracteres no ASCII
La referencia a la licencia es válida.
El paquete tiene autor conocido.
Dependencias
Las dependencias indicadas están instaladas
No hay dependencias circulares.
Todas las referencias del fichero NAMESPACE están listadas como dependencias.
Toda dependencia está listada en NAMESPACE.
Espacios de nombres
El fichero NAMESPACE existe y está bien formado.
Código
No hay caracteres no ASCII en los ficheros fuente.
No hay errores de sintaxis en los ficheros fuente.
No hay dependencias en los ficheros fuente sin declarar en el fichero DESCRIPTION.
Otras comprobaciones sobre el código de R, como no usar funciones fuera del espacio de nombres, no usar funciones obsoletas etc.
Datos
Los ficheros contenidos en el directorio de datos tienen un formato válido.
Los ficheros de datos están comprimidos usando alguno de los algoritmos aceptados.
Test
No hay dependencias sin declarar en los test.
Todos los test en el directorio /test son válidos.
Documentación autogenerada
Los ficheros Markdown están bien formados.
Los nombres y alias declarados en los ficheros Markdown son únicos.
Ninguna línea de un fichero Markdown contiene más de 90 caracteres
Todos los objetos exportados por el paquete están documentados-
Todos los argumentos están documentados.
Todos los ejemplos documentados pueden ejecutarse sin problemas.
El pdf de documentación puede generarse usando LaTeX sin problemas.
Viñetas (documentación manual)
La documentación puede formarse correctamente.
El código R contenido en las viñetas puede ejecutarse correctamente.
Los ficheros de documentación han sido compilados en ficheros *HTML*.

TABLA 5 TABLA RESUMEN DE LOS REQUISITOS PARA QUE UN PAQUETE R ESTÉ BIEN FORMADO

8.2. ESTRUCTURA BÁSICA DEL PAQUETE

La estructura de la aplicación desarrollada está marcada principalmente por lo requerido para generar un paquete de R. Sin embargo, contiene algunas particularidades que comentamos a continuación.

- ***App.R*:** Este fichero, innecesario para el paquete, es el punto de anclaje necesario para despliegue en *shinyapps*, aunque no añade funcionalidad alguna y es innecesario para el paquete.
- ***DESCRIPTION*:** Contiene el resumen de la información del paquete, incluyendo el nombre del paquete, versión, descripción, autores, licencia y las dependencias del paquete.
- ***NAMESPACE*:** Especifica qué funciones y objetos están disponibles para los usuarios del paquete, y qué otros paquetes de R son necesarios para su ejecución. Roxigen2 se encarga de generar este fichero automáticamente.
- ***Renv.lock:*** El fichero generado por el paquete renv que contiene todas las dependencias del proyecto, es decir, contiene información sobre todos los paquetes de R que el proyecto necesita para funcionar, incluyendo la versión exacta de esos paquetes. Esto permite restaurar el estado de todas las dependencias en otro sistema, de forma que facilita el despliegue de la aplicación en cualquier sistema.
- ***/dev:*** Esta carpeta, contiene ejemplos de funciones útiles para los desarrolladores durante todo el ciclo de vida del software. Contiene la lista de funciones necesarias para crear nuevos módulos, testarlos, ejecutar en el entorno de pruebas, añadir dependencias etc.
- ***/R:*** Esta carpeta contiene los ficheros principales de la aplicación. *App.R* llama a *server.R* y *ui.R* que son los módulos principales de la aplicación. *Ui.R* se encarga de construir el *fron-end* de la aplicación, llamando a los diferentes componentes y Server se encarga de generar el código que se encargará de la funcionalidad. Su responsabilidad es instanciar los servidores de cada módulo y establecer la comunicación entre ellos.
    - *core_funct.R*: contiene la lógica de negocio de la aplicación, esto es, las funciones en las que están implementadas las validaciones establecidas por el Real Decreto.
    - *Data.R*: Es un fichero generado simplemente para documentar adecuadamente los ficheros de datos incluidos en el paquete.
    - *golem_*.R*: Ficheros autogenerados por *Golem*. Proporcionan encapsulan funcionalidad exclusiva de JavaScript y *Shiny*.
    - *Mod_*.R*: Ficheros de módulos. Contienen la lógica necesaria para cada módulo.
    - *app_ui.R* y *app_server.R*: Contienen la funcionalidad de la interfaz y el servidor principales.
    - *Globals.R*: Contienen la definición de datos global.
- ***/data:*** Los conjuntos de datos del paquete en formato *RData* o *rda*. En nuestro caso se encuentran las tablas de parámetros recogidas en el Real Decreto, que servirán para los análisis de lotes.

- ***/man:*** La documentación del paquete en archivos *Rd*. En nuestro caso estos ficheros son autogenerados por roxygen2 a partir de los *docstrings* de las diferentes funciones u objetos.
- ***/test:*** Contiene los ficheros de pruebas de la aplicación.
- ***/renv:*** Contiene la infraestructura necesaria para la configuración del paquete *renv*.
- ***/rsconnect:*** Ficheros necesarios para el despliegue de la aplicación.
- ***/vignettes***: Este directorio puede contener tutoriales o documentos explicativos que muestran cómo usar el paquete. En nuestro caso no hemos escrito un manual de uso del paquete.
- ***/docs***: La documentación del paquete del directorio */man* y */vingettes* compilada en un sitio web gracias al paquete *pkgdown*.

8.3. MÓDULOS

Shiny permite dividir una aplicación en unidades funcionales a criterio del programador usando módulos. Esto permite una mejor organización y mayor reutilización del código. Un módulo de *Shiny*, igual que una aplicación completa consta de una parte interfaz y otra parte servidor.

Los identificadores de la interfaz de usuario en *Shiny* son globales, ya que hacen referencia a elementos generados en el *HTML* de la web. Esto hace necesario que, si se desea mantener la modularidad, haya que darles a los elementos un identificador dentro del espacio de nombres del módulo. *Shiny* permite hace esto fácilmente usando la función NS. Una vez hecho esto, los elementos reactivos *$input* y *$output* utilizados en el servidor del módulo apuntarán automáticamente a los elementos de interfaz declarados.

Un módulo puede ser instanciado invocando su UI en la declaración de la interfaz de la aplicación (o de otro módulo) y llamando a su parte servidor de la misma forma. Para ello hace falta pasarle un argumento id único que se utilizará para generar el espacio de nombres de módulo. No es posible reutilizar la misma instancia de un módulo en diferentes partes de la aplicación.

La función servidor de un módulo puede recibir argumentos de entrada y devolver elementos como cualquier otra función de R. Estas salidas deben ser, por su propia naturaleza, reactivas. Esto nos permite comunicar módulos de forma jerárquica de tal forma que un submódulo complejo se trate igual que cualquier otro input u output nativo.

Además de esta forma, todos los módulos de una aplicación tienen acceso a un mismo objeto *Session*, por tanto, es posible hacer operaciones de E/S escribiendo sobre él, ya sea sobre variables reactivas o utilizando una clase R6 para encapsular la funcionalidad. En nuestra aplicación decidimos usar este objeto para comunicar nuestros módulos dado que nos era necesario comunicar módulos jerárquicamente muy alejados.

Aunque al principio del desarrollo esta parecía una aproximación razonable, más adelante nos dimos cuenta de que el resultado de esta aproximación había sido un

acoplamiento mayor de lo deseado de los módulos y una complejidad más alta de la esperada, aunque no llegamos a refactorizar este punto. En una futura refactorización sería aconsejable como mínimo utilizar una clase *R6*.

8.3.1. ESTRUCTURA MODULAR DE LA APLICACIÓN.

Como hemos discutido anteriormente, nuestra aplicación hace uso de esto para dividir el código en módulos. En nuestro caso decidimos que las unidades funcionales que requería nuestra aplicación correspondían a cada una de las páginas de la app.

Así, la página "Resumen", "Análisis de no conformidades", "Análisis de la media" e "Informe" son módulos independientes. Por otro lado, dentro de cada uno de estos módulos se instancia el módulo "*Sidebar*" que, además de contener la interfaz de la barra lateral, donde se encuentran todas las entradas de la app, contiene la lógica necesaria para invocar las funciones que realizan el análisis.

El diseño visual marcó negativamente, en nuestro caso, la estructura interna de nuestra aplicación. *Shiny* nos proporcionaba dos elementos predefinidos para definir la estructura visual de nuestra aplicación fácilmente*: Sidebar layout* y *Navbar layout*.

Sin embargo, el boceto de la interfaz de usuario que realizamos antes de tener suficiente experiencia usando *Shiny* combinaba estos dos elementos (Ilustración 13). Esto no supone ningún problema siempre que se usen en el orden correcto: la barra de navegación como submódulo de la barra lateral.

ILUSTRACIÓN 13 DIVISIÓN VISUAL DE LA PANTALLA PRINCIPAL EN MÓDULOS

En nuestro caso esto suponía un problema puramente estético: nos parecía que la barra de navegación superior debía ocupar todo el ancho de la pantalla y la barra lateral nacer inmediatamente debajo de él. Podríamos haber utilizado *CSS* para solucionar esto y usar una estructura de la aplicación más simple, pero debido a que uno de nuestros objetivos iniciales era evitar utilizar lenguajes fuera de *R* para el desarrollo decidimos simplemente instanciar varias veces la barra lateral, juzgando erróneamente que no supondría un problema grave.

Para ofrecer al usuario una buena experiencia de uso instanciar varias veces la barra lateral requería sincronizar el estado de las diferentes instancias, es decir, modificar visualmente el estado de todos los inputs cuando el usuario modificaba uno.

En *Shiny* esto tenía dos implicaciones. La primera es que era necesario comunicar transversalmente todos los módulos *Sidebar* de nuestra aplicación, y dado que el grueso de los análisis se realizaba sobre el módulo *Sidebar* decidimos usar el objeto *Session* para compartir información entre los módulos en una suerte de *Singleton*.

La segunda implicación venía dada por que *Shiny* no distingue el origen de las modificaciones del estado de los elementos de entrada. Es decir, la actualización de un elemento de entrada lanza un evento sobre su objeto reactivo se actualice desde código o por interacción con el usuario. Eso provocaba que, al actualizar el estado de los inputs por la interacción del usuario de algún otro, se produjera una propagación descontrolada de la reactividad, lanzándose muchos más eventos de los necesarios.

Para subsanar esto el conjunto del *Golemverse* tiene un pequeño paquete llamado *Gargoyle* que permite transformar la reactividad de la aplicación en eventos para facilitar su manejo. El uso de *Gargoyle* nos permitió controlar la reactividad de forma más eficiente.

8.3.2. GARGOYLE

Gargoyle es un paquete del *Golemverse* creado por Collin Fay que permite introducir en una aplicación *Shiny* elementos de la programación orientada a eventos de cara a permitir un flujo más controlado de la reactividad.

Para manejar estos eventos *Gargoyle* nos proporciona una forma de declarar “tipos” de evento en el objeto sesión y dos funciones similares a cualquier otra implementación de la programación por eventos:

- *Trigger*: Lanza un evento del tipo que se desee, este evento se propaga por todos los módulos de la aplicación mediante el objeto *Session*.
- *Watch*: Captura el evento que se desee, ejecutando código de forma condicionada a dicho evento.

A cambio, de ese control más granular de la reactividad *Gargoyle* hizo nuestra aplicación algo más verbosa ya que, como condición para usarlo, nos pareció importante evitar mezclar código puramente reactivo con código manejado por eventos, por tanto, lo implementamos extensivamente en la aplicación.

Diseñamos, de este modo, el flujo de nuestros eventos de forma que los inputs de usuario lanzaran dos tipos de eventos posibles:

1. Actualización de Inputs: Este evento se lanza una vez el usuario ha realizado cualquier input, de cara a que el resto de *Inputs* vinculados puedan actualizarse a su vez.
2. Actualización de Outputs: Este evento se lanza cuando la actualización de una entrada implica la modificación de los resultados, de cara a que los outputs de toda la aplicación puedan actualizarse en consecuencia.

De esta manera, cuando un usuario introduce, por ejemplo, un cambio en la cantidad nominal sobre la que se realiza el análisis, pueden ocurrir dos cosas:

1. Se lanza el evento de actualización de inputs, que es recogido por todas las instancias del cajetín de entrada de la cantidad nominal para actualizar el valor. Estas actualizaciones no generan nuevos eventos, a diferencia de lo que ocurría con la reactividad pura.
2. Si el usuario ya ha introducido los elementos necesarios para realizar el análisis (las muestras, el tamaño del lote…) se calculan los resultados y se lanza el evento de actualización de Outputs. Este evento es recogido por los módulos que obtienen los resultados mediante los datos almacenados en *Session* y actualizan los elementos de salida con los resultados del nuevo análisis.

8.4. TESTING

Para testar adecuadamente la aplicación hemos usado una combinación de pruebas unitarias y de integración. Estas pruebas las hemos gestionado usando la librería *Testthat*, una de las herramientas más comunes en R. Esto nos ha permitido organizar nuestros test en el directorio test de la herramienta y ejecutarlos automáticamente dentro de nuestros flujos de integración continua.

Para las pruebas de integración hemos utilizado *Shinytest2* que, además de integrarse a la perfección en el entorno de *Testthat* nos permite probar de manera sencilla comportamientos complejos como el de la reactividad y el aspecto visual de la aplicación.

8.4.1. TESTTHAT

Para la gestión de pruebas hemos usado Testthat. Esta librería es una de las más estándar de R para realizar pruebas unitarias [21] y gestionar la ejecución de los test automáticamente. Testthat, además, se integra a la perfección con el desarrollo de un paquete, ya que autogenera un fichero *testthat.R* que sirve para probar todo el paquete y es invocado por la instrucción *R CMD Check.*

El componente principal de un test en *testthat* son las aserciones (Fragmento de código 1), que son funciones que comprueban un resultado. Una aserción comienza con el prefijo *expect_*, seguido del resultado esperado. Por ejemplo, *expect_equal(X, Y)* fallará si el valor de *X* e *Y* son diferentes. Existe una gran variedad de estas aserciones como *expect_error, expect_true, expect_warning* etc.

Testthat organiza los test en función del concepto contexto. El contexto en una función de *testthat* es el título que el programador le da al conjunto de aserciones que contiene dicha función, agrupándolas semánticamente. De esta forma, si fallara alguna de las aserciones, *Testthat* nos lo indicaría mediante su contexto. Los test se crean usando la función test_that()que toma un título (o contexto) y con varias funciones de aserción.

```
test_that("get_sample_sizes returns correct results", {
  expect_equal(get_sample_sizes(300, "first"), 30)
  expect_equal(get_sample_sizes(1500, "first"), 50)
  expect_equal(get_sample_sizes(4000, "second"), 80)
})
```

FRAGMENTO DE CÓDIGO 1 EJEMPLO DE TEST UNITARIO

Los test se pueden realizar manualmente llamando a la función *test_file* pero una de las grandes ventajas de *Testthat,* como hemos comentado, es que se integra con el comando *devtools::test()* y *devtools::check(),* que ejecutan todos los archivos de prueba en el directorio de test del paquete. Esto nos facilita la tarea a la hora de realizar scripts de integración continua.

En nuestro caso hemos usado extensivamente *Testthat* para realizar test unitarios sobre las funciones principales de nuestra aplicación y para organizar los test de integración creados con shinytest2.

8.4.2. PRUEBAS DE INTEGRACIÓN CON SHINYTEST2

Existen dos librerías para realizar pruebas de integración sobre aplicaciones S*hiny* de forma sencilla: *Shinytest* y *Shinytest2*. Ambas se basan en el concepto de *snapshot*.

La idea básica de podemos grabar (o programar) una serie de interacciones con la aplicación, almacenar el resultado visual de estas interacciones y luego reproducirlas para comprobar si la aplicación aún funciona como se espera.

Durante la reproducción, la librería hará capturas del estado de la aplicación (o almacenará el código HTML generado) y las comparará con las capturas (o *snapshots*) de referencia almacenadas. Cuando estas comprobaciones fallan, *Shinytest* permite al programador almacenar nuevas capturas en caso de que el programador lo considere.

De este modo, y de forma similar a las aserciones de *testthat*, estas librerías nos proporcionan su propio conjunto de aserciones, entre ellas:

- *expect_values():* Comprueba que los valores almacenados en *input* y *output* se correspondan con lo esperado.
- *expect_download():* Comprueba que se descarga un fichero igual al esperado.
- *expect_text():* Comprueba el texto seleccionado de la pantalla.
- *expect_html():* Comprueba el HTML generado.
- *expect_screenshot():* Comprueba el estado de la aplicación visualmente a partir de una captura de pantalla.

El problema de estas pruebas, especialmente las capturas de imágenes, es que pueden producir falsos negativos en casos de cambio de sistema, resolución o cambios mínimos en la aplicación.

Originalmente nos pareció que la forma de trabajar con *Shinytest* era muy adecuada para realizar pruebas de integración de la herramienta de forma sencilla, ya que incluso se te

proporciona una interfaz *no-code* que permite autogenerar código de pruebas. Sin embargo, *Shinytest* dependía de una librería, *webdriver* [24] y *PhantomJS* para replicar las acciones a testar. Dado que *PhantomJS* lleva sin mantenerse desde 2017 había ciertas funcionalidades de la aplicación (como todas las referencias a clases de *Bootstrap 5*) que no podían probarse correctamente con la librería.

Debido a esto, y después de buscar algunas alternativas descubrimos que recientemente se había publicado una nueva versión de la librería llamada Shinytest2 [25] que ofrecía una funcionalidad similar, pero usando *chromote* como driver (Fragmento de código 2).

```
test_that("{shinytest2} recording: SecondAnalysisWithInput", {
  app <- AppDriver$new(variant = platform_variant(), name =
"SecondAnalysisWithInput",
      height = 969, width = 1619)
  app$set_inputs(`overview_1-sidebar_1-LabeledQuantity` = 37)
  app$set_inputs(`overview_1-sidebar_1-LabeledQuantity` = 33)
  app$upload_file(`overview_1-sidebar_1-first_sample` = file.path("..",
"files", "input.txt"))
  app$set_inputs(`overview_1-sidebar_1-first_sample_column` = "Volumen")
  app$click("overview_1-sidebar_1-do")
  app$upload_file(`overview_1-sidebar_1-second_sample` = file.path("..",
"files", "input.txt"))
  app$set_inputs(`overview_1-sidebar_1-second_sample_column` = "Volumen")
  app$click("overview_1-sidebar_1-do")

  app$expect_screenshot()
  app$expect_values()
})
```

FRAGMENTO DE CÓDIGO 2 EJEMPLO DE TEST DE INTEGRACIÓN USANDO SHINYTEST2

8.5. DOCUMENTACIÓN DE FUNCIONES

Para documentar las funciones desarrolladas, y como parte de los requisitos para el desarrollo de un paquete de R completo, hemos usado Roxigen2.

8.6. ROXYGEN2

Roxigen2 es una librería que facilita la documentación de los paquetes de *R*. En lugar de escribir archivos de documentación *Rd* manualmente, nos permite generarlos en base a los comentarios de documentación dentro de las funciones (o *docstrings*). Esto facilita la actualización de la documentación cuando se modifica una función. Un ejemplo de la documentación generada por *roxygen2* se encuentra en la Ilustración 14.

Además, *roxygen2* nos permite incluir ejemplos dentro de la documentación, que luego serán comprobados al usar *R CMD Check*, lo que obliga a que el código de ejemplo de la documentación esté siempre actualizado.

Roxygen2 usa un formato específico para sus comentarios mediante etiquetas (Fragmento de código 3), para que más adelante puedan ser procesados. De esta forma,

la etiqueta *@param* se utiliza para describir los parámetros de una función, *@return* para describir lo que devuelve y *@examples* para proporcionar ejemplos de uso.

Además, *Roxygen2* cumple un papel fundamental dentro de la estructura de un paquete R mediante las etiquetas *@import* y *@export*. Estas etiquetas marcan, en el caso de *@export*, las funciones destinadas a los usuarios del paquete, el equivalente a las funciones públicas de otros lenguajes. En el caso de *@import* marca las dependencias de esa función concreta.

Adicionalmente, hemos usado *pkgdown* [26], una librería auxiliar, que nos permite generar un sitio web automáticamente en base a la documentación generada por *roxygen2* y los ficheros *RMD* creados en el directorio */vingettes*.

```
#' Get Sample Sizes
#'
#' Calculates the required sample size for a given batch size and
analysis type, based
#' on a reference table of sample sizes.
#'
#' @param batch_size A numeric value indicating the batch size for which
the sample size
#' is to be calculated.
#'
#' @param analysis A string value indicating the type of analysis for
which the sample size
#' is to be calculated. Must be either "first" or "second".
#'
#' @return A numeric value indicating the required sample size for the
given batch size
#' and analysis type. If the batch size is outside the range of the
reference table, or if
#' the analysis parameter is not valid, the function returns an error
message.
#'
#' @export
#'
#' @examples
#' get_sample_sizes(300, "first")
#' get_sample_sizes(1500, "second")
get_sample_sizes <- function(batch_size, analysis) {
  if (!analysis %in% c("first", "second", "mean")) {
    stop("Analysis parameter must be either 'first' 'second' or 'mean'.")
  }

  sample_col <- paste0(analysis, "_analysis_sample")

  limits <- sample_sizes %>%
    filter(batch_size >= .data$batch_size_min, batch_size <=
.data$batch_size_max) %>%
    pull(sample_col)

  if(length(limits) > 0) {
    limits %>% as.numeric
  } else {
    stop("Batch size is outside the range of possible values.")
  }
```

```
}
```

FRAGMENTO DE CÓDIGO 3 FUNCIÓN DOCUMENTADA USANDO ROXIGEN2.

Get Sample Sizes

Description

Calculates the required sample size for a given batch size and analysis type, based on a reference table of sample sizes.

Usage

```
get_sample_sizes(batch_size, analysis)
```

Arguments

batch_size	A numeric value indicating the batch size for which the sample size is to be calculated.
analysis	A string value indicating the type of analysis for which the sample size is to be calculated. Must be either "first" or "second".

Value

A numeric value indicating the required sample size for the given batch size and analysis type. If the batch size is outside the range of the reference table, or if the analysis parameter is not valid, the function returns an error message.

Examples

Run examples

```
get_sample_sizes(300, "first")
get_sample_sizes(1500, "second")
```

ILUSTRACIÓN 14 DOCUMENTACIÓN GENERADA USANDO ROXYGEN2

8.7. RESULTADO

Concluido el desarrollo, hemos creado una aplicación *Shiny* con siete páginas diferentes:

- Resumen (Ilustración 15): Aquí puede observarse una vista general del análisis, de forma que un usuario avanzado pueda limitarse a realizar el análisis desde la primera página, sin tener que navegar más por la aplicación. Además, el texto descriptivo permite a un usuario novato usar la aplicación sin necesidad de utilizar ningún manual. Una última opción permite a los usuarios descargar un pequeño informe del análisis realizado.

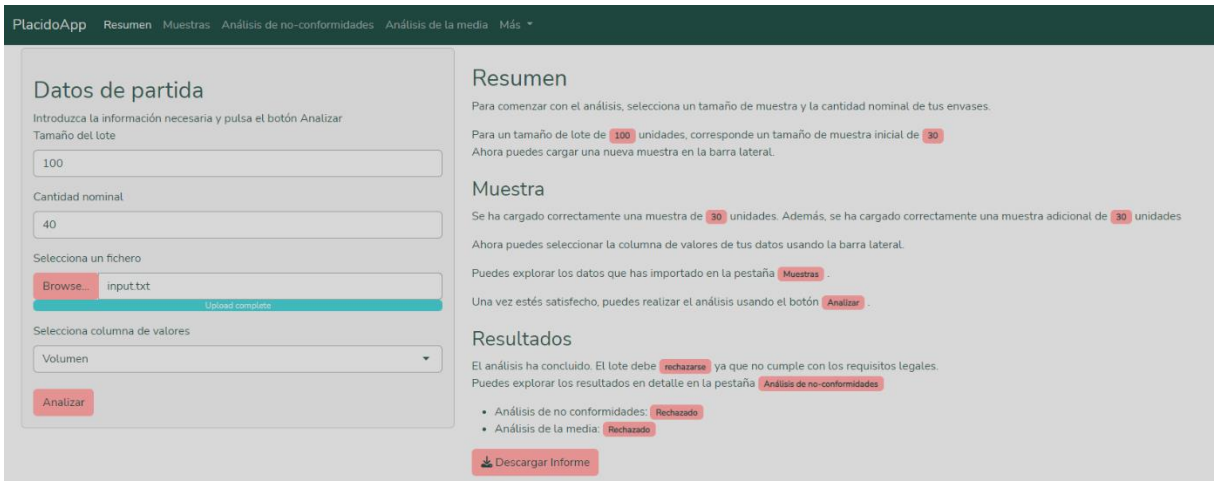


ILUSTRACIÓN 15 HOJA RESUMEN DE LA APLICACIÓN.

- Muestras: Esta hoja permite al usuario explorar las muestras cargadas en la aplicación.
- Análisis de no-conformidades: Esta hoja permite al usuario explorar el resultado del análisis de no-conformidades, con un gráfico de control ilustrativo y la muestra categorizada por envases aceptables, no-conformes o rechazados.
- Análisis de la media (Ilustración 16): Esta hoja permite al usuario explorar el resultado del análisis de la media, con un gráfico ilustrativo y una explicación textual.

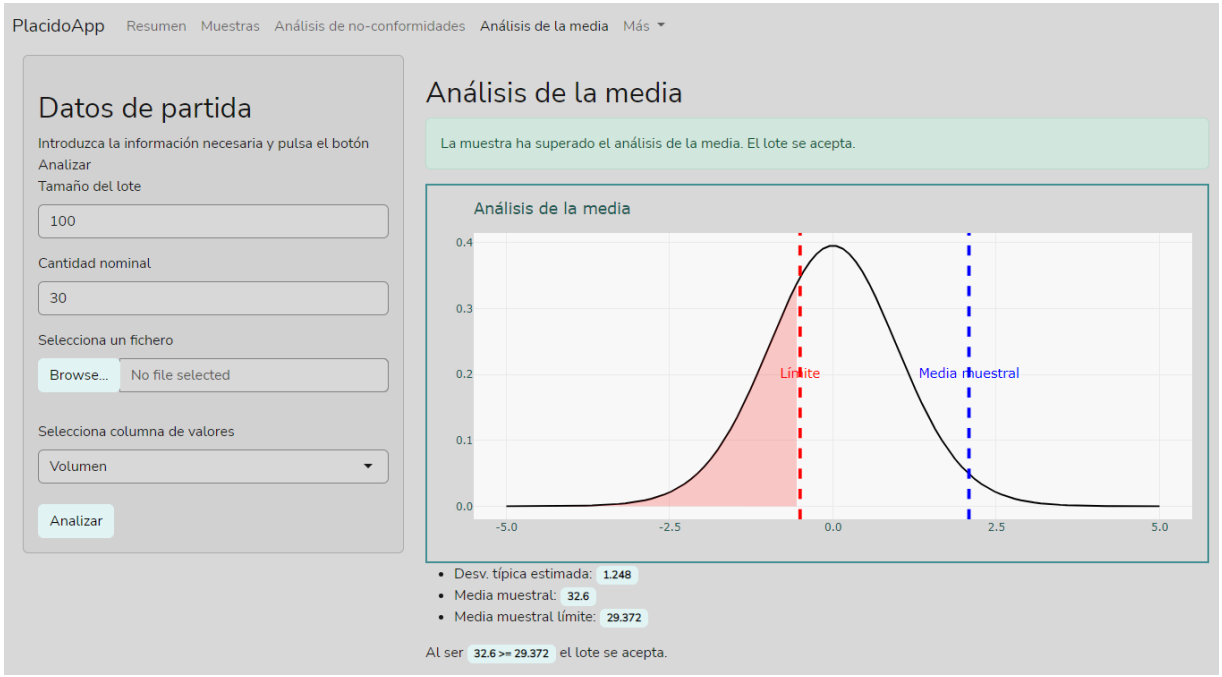


ILUSTRACIÓN 16 ANÁLISIS DE LA MEDIA CON EL TEMA DE ALTO CONTRASTE APLICADO

- Más: En esta sección se encuentran las siguientes hojas auxiliares:

- Datos de ejemplo: Que permite al usuario cargar datos de ejemplo en la aplicación con tres resultados diferentes.
- Ajustes visuales: Que permite al usuario cambiar al tema visual de alto contraste.
- Más información: Que contiene información adicional sobre la aplicación.

En estas siete páginas se encuentran implementadas 11 de las 14 historias de usuario definidas inicialmente. Además, se han implementado 4 de las 5 historias técnicas. La implementación completa de los requisitos de accesibilidad establecidos por el *W3C* no ha sido posible dado que requería del uso de JavaScript fuera de las opciones nativas de *Shiny*.

9. CONCLUSIONES

A lo largo de este trabajo hemos establecido una metodología para el desarrollo de una aplicación *Shiny* en línea con los estándares actuales de la industria del desarrollo Software, aplicable a cualquier proyecto.

Hemos adaptado Scrum, para adecuarse a un desarrollo pequeño, con pocos programadores implicados, pero respetando en todo momento los principios más importantes del Manifiesto Ágil. Por otro lado, hemos demostrado cómo integrar el control de versiones, la integración y la entrega continuas en la creación de una aplicación *Shiny* de forma lo más simple, utilizando herramientas estándar y de código abierto.

Por otro lado, hemos explorado algunos de los paquetes del ecosistema Shiny y sus posibilidades. Hemos usado *bslib* para crear un tema visual para nuestra aplicación. Hemos utilizado *shinytest2* y *testthat* para crear pruebas unitarias y de integración. Y hemos utilizado *gargoyle* para controlar la reactividad a bajo nivel constriñéndola a un paradigma orientado a eventos.

Además, hemos estudiado cómo utilizar *Golem* para desarrollar una aplicación Shiny como un paquete de pleno derecho, beneficiándonos de la estructura que otorga a la aplicación, de cara a realizar desarrollos más complejos.

Con todo, hemos presentado una aplicación web desarrollada completamente en R, que permitiría a una empresa productora o distribuidora de productos envasados cumplir con todos los requisitos legales presentes en el Real Decreto 1801/2008. Además, debido a la estructura como paquete R, hemos puesto a disposición de los usuarios las funciones que realizan dicha tarea de forma independiente, lo que les permitiría desarrollar sus propias soluciones usándolas.

Creemos por tanto que *Shiny* ofrece una gran oportunidad a todo tipo de analistas de datos para crear aplicaciones web usando lenguajes que le son familiares, hecho que previsiblemente se verá acentuado con el reciente anuncio de una librería *Shiny* para Python. Y que *golem* les ofrece a estos usuarios una forma más estructurada de desarrollar aplicaciones, facilitando la creación de proyectos mucho más complejos.

Pese a esto, el equilibrio es difícil. Algunas de las limitaciones de *Shiny*, como la seguridad, no tienen una solución clara y otras, como la personalización, pueden atacarse, pero aumentado significativamente la dificultad, perdiendo en el proceso la agilidad que caracteriza a un desarrollo de este estilo. Así, la reactividad, que ofrece una solución elegante y sencilla para crear algo de interactividad en las aplicaciones, puede convertirse en impráctica rápidamente, y una librería como *gargoyle*, que aborda esta problemática, añade, sin embargo, una capa de complejidad extra a la aplicación.

Más aún, muchas de estas librerías están aún en una fase temprana de su desarrollo, lo que puede provocar inestabilidad en su comportamiento, lo que puede limitar seriamente su uso en aplicaciones con enfoque comercial.

9.1. LECCIONES APRENDIDAS

Durante la creación de esta aplicación, hemos ido adquiriendo una serie de valiosas lecciones de mejora de cara a un futuro desarrollo o la continuación de este:

1) El desarrollo de un paquete completo conlleva un trabajo importante, que es necesario valorar en el alcance del proyecto. Para ciertas aplicaciones puede no ser costo eficiente.
2) De igual modo, la personalización supone un coste importante sobre el total de la aplicación. Es recomendable analizar bien si es necesario salirse de las opciones predefinidas que ofrece *Shiny*.
3) Por esto mismo, si un equipo va a realizar varias aplicaciones similares, puede ser interesante invertir esfuerzo en crear widgets personalizados utilizando JavaScript, de forma que se obtenga una librería propia de elementos.
4) El uso de *renv* para fijar las dependencias exactas de la aplicación facilita enormemente despliegue de una aplicación *Shiny*.
5) Si el diseño de la interactividad no es completamente lineal es muy recomendable convertir la reactividad en eventos usando *Gargoyle*, no sólo facilita el desarrollo, sino que aumenta en gran medida la eficiencia.
6) El diseño de la interfaz debe hacerse teniendo en cuenta las limitaciones y los diseños de página ofrecidos por *Shiny*, esto evita en gran medida que la comunicación entre los diferentes módulos se complique.
7) La disciplina en cuanto al seguimiento de la metodología es muy importante para obtener los beneficios que esta implica.

9.2. TRABAJO FUTURO

Aunque la aplicación *Shiny* desarrollada integra la mayoría de las historias de usuario definidas, debido a limitaciones de tiempo, dos historias nos parecen muy interesantes de cara a un trabajo posterior:

1) La posibilidad de crear un plan de muestreo personalizado: Ya que el Real Decreto permite crear planes de muestreo personalizados, siempre que sean equivalentes a los recogidos en la norma, aportaría gran valor a los usuarios esta funcionalidad.
2) La posibilidad de almacenar un análisis mediante la generación de *URLs*: El desarrollo de esta funcionalidad nos permitiría explorar las posibilidades de ruteo que ofrece *Shiny*.

Por otro lado, hay otras dos vías por las que este trabajo podría ser ampliado en el futuro:

1) Levantar la restricción de uso de lenguajes de programación web y refactorizar la aplicación usando trozos de código *JavaScript* y *CSS*, de forma que exploremos cómo deben incluirse esos códigos para no perjudicar la mantenibilidad del proyecto.
2) Subir el paquete a *CRAN* donde pueda ser utilizado más fácilmente por la comunidad de R.

10. ANEXOS

Adjuntos a esta memoria se entregan:

- Fichero *code.zip*: Con los archivos fuente del proyecto creado.
- Fichero *Manual_de_usuario.pdf*: Con un pequeño manual de uso de la aplicación.
- Fichero *RealDecreto.pdf*: Con el Real Decreto 1801/2008 cuyos procesos se implementan en este trabajo.

Además, accesibles públicamente se encuentra:

- La aplicación desplegada en shinyapps.io: https://dleo.shinyapps.io/placidoproject/
- El repositorio de GitHub con el código fuente: https://github.com/DavidALeo/PlacidoProject
- La documentación generada como sitio web alojada en GitHub Pages: https://davidaleo.github.io/PlacidoProject/
- El Real Decreto 1801/2008, de 3 de noviembre, por el que se establecen normas relativas a las cantidades nominales para productos envasados y al control de su contenido efectivo: https://www.boe.es/eli/es/rd/2008/11/03/1801

11. Referencias

[1] J. Li, B. Cui, Y. Dai, L. Bai y J. Huang, «BioInstaller: a comprehensive R package to construct interactive and reproducible biological data analysis applications based on the R platform,» *PeerJ,* 2018.

[2] P. Kasprzak, L. Mitchell, O. Kravchuk y A. Timmins, «Six Years of Shiny in Research - Collaborative Development of Web Tools in R,» *The R Journal,* vol. 12, nº 2, pp. 155-162, 2020.

[3] W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert y B. Borges, shiny: Web Application Framework for R, 2022.

[4] B. Schloerke, «reactlog,» 2019.

[5] Ministerio de la Presidencia, Real Decreto 1801/2008, de 3 de noviembre, por el que se establecen normas relativas a las cantidades nominales para productos envasados y al control de su contenido efectivo., Madrid: BOE, 2008.

[6] E. P. E. Y. E. C. D. L. U. EUROPEA, DIRECTIVA 2007/45/CE DEL PARLAMENTO EUROPEO Y DEL CONSEJO, Bruselas: Diario Oficial de la Unión Europea, 2007.

[7] softwarepath, «ERP Report,» softwarepath, 2022.

[8] M. Haddara y A. Elragal, «ERP adoption cost factors identification and classification: a study in SMEs,» *International Journal of Information Systems and Project Management,* vol. 1, nº 2, pp. 5-21, 2022.

[9] Bizerba, «Web corporativa de Bizerba,» [En línea]. Available: https://www.bizerba.com/es/es/products/brain2. [Último acceso: 10 06 2023].

[10] Dibal, «Web corporativa de Dibal,» [En línea]. Available: https://www.dibal.com/es/software-para-equipos-de-control-de-peso-y-clasificacion-des. [Último acceso: 10 06 2022].

[11] ASM, «Web corporativa de ASM,» [En línea]. Available: https://asm.es/modulo-cce-la-solucion-orientada-a-facilitar-el-cumplimiento-de-la-normativa-de-control-de-contenido-efectivo-rd-1801-2008/. [Último acceso: 10 06 2022].

[12] Giropes, «Web corporativa de Giropes.,» [En línea]. Available: https://www.giropes.com/es/noticias/software-giscale-acredita-el-envasado-efectivo-de-productos-44/. [Último acceso: 10 06 2022].

[13] DAII S.L., «Web corporativa de DAII S.L.,» [En línea]. Available: https://www.daiisl.com/ow.aspx#:~:text=OpenWeight%20es%20una%20soluci

%C3%B3n%20integral,simplifica%20y%20facilita%20su%20ejecuci%C3%B3n. [Último acceso: 10 06 2022].

[14] Dirección General de Industria, Energía y Minas. CONSEJERÍA ECONOMÍA, EMPLEO Y HACIENDA. Comunidad de Madrid, «Campañas de inspección realizadas por la Comunidad de Madrid,» 2018.

[15] Scrum.org, «16th Annual State of Agile Report,» 2022.

[16] W3C, «Web Content Accessibility Guidelines (WCAG) 2.0,» 2008.

[17] D. ÖZKAN y A. MISHRA, « Agile Project Management Tools: A Brief Comprative View.,» *Cybernetics and Information Technologies,* vol. 19, nº 4, pp. 17-25, 2019.

[18] Atlassian, «Web corporativa de Atlassian,» [En línea]. Available: https://www.atlassian.com/es/company. [Último acceso: 12 06 2023].

[19] S. Chacon y B. Straub, Pro git, Apress, 2014.

[20] C. SIEVERT, «Interactive web-based data visualization with R, plotly, and shiny.,» CRC Press, 2020.

[21] H. Wickham y J. Bryan, R Packages (2º Edición), O'Reilly, 2023.

[22] C. Fay, S. Rochette, V. Guyader y C. Girard, Engineering Production-Grade Shiny Apps, Chapman & Hall/CRC, 2022.

[23] Appsilon, «shiny.i18n: Shiny Applications Internationalization,» [En línea]. Available: https://CRAN.R-project.org/package=shiny.i18n.

[24] A. Hidayat, G. Csárdi, G. Torok, I. De Marino y R. Gieseke, «'WebDriver' Client for 'PhantomJS',» *CRAN,* 2021.

[25] B. Schloerke, «shinytest2: Testing for Shiny Applications,» 2023.

[26] H. Wickham, J. Hesselberth y M. Salmon, «pkgdown: Make Static HTML Documentation for a Package,» 2022.

[27] H. Wickham, Mastering Shiny, O'Reilly Media, 2020.