UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario de Ingeniería de Software

31105151 - Trabajo Fin de Máster

“Modelo de trabajo y buenas prácticas para un proyecto de migración de una aplicación bancaria en entorno Mainframe”.

Alumna: María del Mar Martín Gázquez

Tutora: María Magdalena Arcilla Cobián

Curso 2020/21

Convocatoria de junio 2021

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario de Ingeniería de Software

31105151 - Trabajo Fin de Máster

"Modelo de trabajo y buenas prácticas para un proyecto de migración de una aplicación bancaria en entorno Mainframe".

Trabajo Fin de Máster Tipo B

Alumna: María del Mar Martín Gázquez

Tutora: María Magdalena Arcilla Cobián

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 15/06/2021

Quién suscribe:

Autora: María del Mar Martín Gázquez D.N.I/N.I.E/Pasaporte.:

Hace constar que es la autora del trabajo:

Modelo de trabajo y buenas prácticas para un proyecto de migración de una aplicación bancaria en entorno Mainframe.

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid
Tel: 91 398 89 10
Fax: 91 398 89 09
www.lsi.uned.es

Resumen

Las características particulares de los proyectos de migración hacen que no sea sencillo llevarlos a buen término siguiendo las metodologías y buenas prácticas que sí pueden funcionar en nuevos desarrollos o en el mantenimiento de aplicaciones. En una migración es necesario, además de construir un producto de acuerdo con unas especificaciones iniciales, seguir el ritmo de evolución de la aplicación original y ofrecer las mismas funcionalidades y el mismo nivel de calidad que ésta desde el primer día de la puesta en producción. El objetivo de este TFM es estudiar las metodologías y buenas prácticas de gestión, desarrollo y migración más comunes, analizar las peculiaridades propias de un proyecto de migración en entorno Mainframe y hacer una propuesta que contribuya al éxito de estas migraciones, evitando así que esto último dependa casi exclusivamente de la experiencia y las dotes organizativas del jefe de proyecto.

Executive summary

Migration projects exhibit specific characteristics that do not exist in methodologies and best practices currently employed for new development or maintenance of applications, making it a challenge to establish working guidelines. A migration consists of not only satisfying the initial specifications, but also maintaining the evolution, features and quality of the original product from day one. The objective of this TFM is to study the most common management, development and migration methodologies and best practices with a particular focus on a Mainframe environment and construct a proposal aimed at improving the success rate of migrations without the need to rely exclusively on the experience and skills of a project manager.

Lista de palabras clave

Migración, Mainframe, Aplicación legada, Aplicación migrada, Sistema legado, Metodología, Modelo de trabajo.

Contenido

Lista de figuras y tablas de contenido

1. INTRODUCCIÓN

1. Introducción

El ámbito bancario fue uno de los primeros en integrar el uso de la informática automatizando cálculos y procesos que hasta entonces se habían venido haciendo de manera manual. Desde el nacimiento de los primeros CPD en la década de los ’60 del siglo XX hasta la actualidad se han desarrollado aplicaciones que implementan la cada vez más compleja operativa bancaria, complejidad que se ve acentuada por la interdependencia existente entre ellas.

Ya sea por evolución tecnológica, por estrategia empresarial o por cualquier otro imperativo que les afecte (Brito, García and Meira, 2010), algunas de estas aplicaciones bancarias deben convertirse a una nueva que mantenga toda su funcionalidad pero que utilice distintos elementos técnicos, como pueden ser un nuevo lenguaje de programación, otra base de datos o la integración dentro de una arquitectura. Aparecen entonces los proyectos de migración, cuyas particulares características hacen complicado llevarlos a cabo bajo modelos y metodologías utilizadas con éxito en otro tipo de desarrollos.

El primer condicionante con el que se encuentra un proyecto de migración es que la nueva aplicación no debe perder ninguna de las funcionalidades que ofrece la antigua hasta el mismo momento en el que se produce la transición de una a otra, ya que el usuario final no debe ver mermado el servicio que venía recibiendo hasta ese momento. Para él, la migración debe ser un proceso totalmente transparente. Esto hace que, salvo que parte de la aplicación origen haya quedado obsoleta antes del inicio de la migración, no quede margen para la negociación sobre el alcance mínimo del proyecto. Significa también que, si se producen cambios funcionales en la aplicación original durante la construcción de la nueva, éstos deben repercutirse igualmente en el nuevo desarrollo. O, dicho de otra manera, las tareas de desarrollo se solapan con las de mantenimiento del propio software que se está construyendo. Cabría pensar en la posibilidad de “congelar” la aplicación origen mientras se desarrolla aquella que la sustituirá unos meses más adelante, pero que este sea el escenario de trabajo es altamente improbable (Teppe, 2009): los cambios en la funcionalidad de las aplicaciones no suelen provenir del capricho de un usuario o de un responsable informático, sino de la necesaria evolución de la normativa bancaria que va adaptándose a las condiciones económicas y sociales de su ámbito de

aplicación. Cuanto más largo sea el tiempo dedicado a la migración, más probabilidades hay de que aparezcan este tipo de cambios.

Otro problema habitual en los entornos bancarios es la escasez de documentación o, más correctamente, la falta de actualización de aquella que se tiene disponible. En estas circunstancias, la mejor fuente de información acaba siendo la propia aplicación que se quiere migrar y la toma de requisitos se transforma en un estudio del código fuente que se desea sustituir. El mayor inconveniente que esto presenta es la dificultad de detectar el posible código muerto existente en los programas, lo que puede suponer la implementación de funcionalidades obsoletas en la nueva aplicación que se construye.

La alta interacción entre aplicaciones que se da en el ambiente bancario también añade complejidad al proceso de migración. Habitualmente los aplicativos bancarios se relacionan entre sí, ya sea enviando o recibiendo notificaciones, intercambiando información, o dando paso a la ejecución planificada de determinados procesos. La migración de una aplicación debe ser lo más silenciosa posible para el conjunto de aquellas con las que se relaciona. Esto es, se debe minimizar el impacto exterior, lo que puede ocasionar restricciones en el diseño e incluso en la implementación de un determinado requisito funcional.

Otro aspecto relativo a la interacción entre las aplicaciones es su bidireccionalidad. Como ya se ha comentado, cuando se migra una aplicación se ven afectadas también aquellas con las que se relaciona, pero de la misma manera, un cambio en esas últimas repercute en aquella que se migra. Ya sea en uno u otro sentido, esta dependencia es directamente proporcional al número de aplicaciones afectadas, y cuantas más interrelaciones existan, mayor dificultad habrá para implementar cambios de alto impacto.

Entrando ya en el terreno de la implantación, otro problema que se afronta es la dificultad de hacerlo de manera escalonada o por fases, tendiendo casi siempre a un modelo big-bang. La dificultad viene impuesta por condicionantes como la fuerte interrelación entre las funcionalidades que conforman la aplicación, la incompatibilidad técnica entre la aplicación nueva y la original, y (como se comentaba al inicio de este capítulo) la obligatoriedad de que en la fecha de entrada en producción se cuente con todas las funcionalidades de las que se venía disfrutando hasta la fecha.

Como último punto a añadir a las dificultades que presenta un proyecto de migración puede mencionarse el de las mejoras de última hora propuestas por el usuario final aprovechando que se va a construir una aplicación nueva. Este usuario final es el que da el visto bueno a la aplicación y sin su aprobación ésta no puede llevarse a producción. En ocasiones ocurre que no se solicita la participación de este usuario hasta el momento en el que se le entrega el nuevo producto para sus pruebas, en otras se intenta contar con él desde las fases iniciales, pero por su resistencia al cambio no toma interés hasta el momento previo de la entrada en producción. Ya sea por uno u otro motivo las sugerencias y peticiones de cambio pueden llegar con escasa antelación o con menos plazo para su implementación del que hubiera sido deseado.

Todo lo que se ha comentado hasta el momento redunda en la planificación del proyecto y en el establecimiento de su fecha de entrada en producción. Un jefe de proyecto que lidere la migración de una aplicación debe hacer frente a todas estas circunstancias y sacar adelante el trabajo sin demoras ni sobrecostes. En numerosas ocasiones esto tendrá que hacerlo valiéndose de su experiencia previa y aprovechando al máximo sus capacidades personales de gestión de equipos, planificación de tareas, estimación de esfuerzos, etc. lo que le convierte en una pieza clave para la consecución de los objetivos marcados.

Este Trabajo Fin de Máster (TFM) aporta un modelo de trabajo para proyectos de migración de grandes aplicaciones que repercutirá en una mejor planificación y estimación de recursos en este tipo de proyectos al conocer de antemano las etapas que deben cumplirse y los factores que afectan al proyecto a lo largo de todo su desarrollo. El TFM se centra en el proceso de desarrollo en sí, por lo que no se incluye como fase ni como proceso premigración el estudio y valoración de criterios objetivos para determinar la conveniencia o no de embarcarse en el proyecto de migración. Es decir, que el modelo de trabajo que aquí se propone comienza una vez que ha finalizado ese estudio de viabilidad y se decide llevar adelante la migración. Se centra en el trabajo que debe llevarse a cabo una vez tomada la decisión de modernizar una determinada aplicación legada y contempla desde los estudios preliminares del entorno y de la aplicación a transformar hasta la puesta en producción de la nueva que se construye.

Además, dado que la tecnología es un ámbito en continua evolución, es de esperar que los proyectos de migración sigan existiendo y que las conclusiones

que se extraigan de este trabajo permanezcan vigentes durante un largo periodo de tiempo.

1.1. Objetivos del TFM

Vistas las dificultades inherentes a la migración de aplicaciones y el diferente escenario que se dibuja con respecto al que corresponde a un nuevo desarrollo, este TFM tiene como objetivo principal encontrar un modelo de trabajo aplicable a los proyectos de migración para que, con la ayuda de herramientas construidas ad hoc y otras generales, su éxito no dependa casi exclusivamente de la experiencia y las dotes organizativas del jefe de proyecto.

Para conseguir este propósito se plantean los siguientes objetivos secundarios:

1. Estudiar las metodologías de migración ya existentes (sean o no específicas de entornos Mainframe).
2. Seleccionar y analizar estándares, ciclos de vida y modelos de gestión aplicables al ámbito bancario Mainframe.
3. Determinar las fases y tareas de las que se compone el modelo de trabajo para un proyecto de migración de una aplicación bancaria.
4. Presentar un caso de estudio en el que se aplique el modelo de trabajo propuesto haciendo uso de herramientas tanto de creación propia como de uso general.

1.2. Estructura del TFM

Este TFM comienza con un primer capítulo de introducción a la temática que se aborda (Capítulo 1), exponiendo las dificultades específicas que presentan los proyectos de migración y los factores que deben tenerse en cuenta a la hora de llevar a cabo su ejecución (determinación del alcance, interacción con otras aplicaciones, evolución de la aplicación origen, etc.). También dentro de este primer capítulo se especifican los objetivos principales y secundarios que se quieren conseguir con la elaboración del TFM, así como la estructura que lo compone.

El siguiente capítulo (Capítulo 2) se dedica al planteamiento del problema y en él se expone la problemática que da lugar a la elaboración de este TFM.

En el tercer capítulo (Capítulo 3) se aborda el estado de la cuestión. Se recopilan algunos modelos, estándares, metodologías y buenas prácticas utilizadas en la actualidad y aplicables al ámbito bancario con el objetivo de dar a conocer en qué consisten y cómo manejan los factores que afectan al éxito de los proyectos. En el capítulo cuarto (Capítulo 4), dedicado a la resolución, se presenta el modelo de migración que se ha diseñado y que se adapta a las circunstancias particulares de los proyectos de migración Mainframe. En él se establecen tanto las fases que se deben seguir como las actividades que quedan englobadas dentro de cada una de ellas. A continuación, en el capítulo quinto (Capítulo 5), se exponen los problemas encontrados en el desarrollo de los proyectos de migración Mainframe para cada una de las etapas o fases propuestas y cómo el nuevo modelo de trabajo aporta soluciones a los mismos. El capítulo sexto (Capítulo 6) está dedicado a la presentación de las conclusiones y a la propuesta de líneas futuras de investigación dentro del ámbito del TFM. El trabajo finaliza con la bibliografía (Capítulo 7), que incluye la recopilación de referencias consultadas para la elaboración de este TFM, el listado de siglas, abreviaturas y acrónimos (Capítulo 8) y un anexo dedicado a los formularios desarrollados para este trabajo fin de máster (Capítulo 9).

2. PLANTEAMIENTO DEL PROBLEMA

2. Planteamiento del problema

Las empresas de consultoría tecnológica estiman que existen aproximadamente entre 180 y 200 mil millones de líneas de código escritas en lenguajes como COBOL, FORTRAN, PL/1 o RPG que se mantienen todavía en uso (Dorninger, Moser and Pichler, 2017) soportando el 75% de las transacciones. Por mencionar un ejemplo concreto, la Seguridad Social de EEUU cuenta con 60 millones de líneas COBOL todavía en activo, cifra similar a la de su sistema de recaudación de impuestos (IRS[1]) que además suma unos 20 millones de líneas codificadas en ensamblador (Charette, 2020).

Estos sistemas corren principalmente en entornos Mainframe y se han mantenido en funcionamiento hasta la actualidad por dos motivos principales (Charette, 2020):

- No han dado ningún problema, es decir, han estado cumpliendo con su cometido sin generar ningún tipo de incidencia.
- Las entidades propietarias no han considerado oportuno o no han podido afrontar el gasto y el riesgo que supone su transformación en un sistema más moderno.

Las aplicaciones que componen estos sistemas conforman por lo general el núcleo principal del negocio de las organizaciones (Althani, 2017). Pueden considerarse la columna vertebral de la información y un fallo en ellas tendría un importante impacto en el negocio. Son especialmente críticas y no pueden estar inoperativas durante un largo periodo de tiempo (Bisbal, 1999), algo que podría ocurrir si no fuera posible compatibilizar su funcionamiento con la tarea de migración.

Sin embargo, inconvenientes como la dificultad que presentan para el escalado o no poder integrarlas de manera sencilla con otros sistemas de la misma instalación (Althani, 2017), o la carencia de profesionales especializados y la falta de documentación (Althani, 2017) pueden acabar convirtiéndolas en sistemas legados y ser por tanto candidatas para ser incluidas en un proceso de transformación.

En cuanto a qué es un sistema legado, la definición más sencilla es la propuesta en el artículo titulado “No one notices the creaky software systems that run the

[1] Internal Revenue Service

world— until they fail[2]", publicado en el número de septiembre de 2020 de la revista Spectrum, del IEEE. En él se afirma que un sistema legado es aquél que, siendo crítico para una organización, queda desactualizado en alguno de sus aspectos (Ej.: funcionalidad, lenguaje de programación, interfaz de usuario, base de datos, etc.). Si esto ocurre, empiezan a sufrirse consecuencias como un incremento en los costes de mantenimiento, una todavía mayor limitación en la capacidad de escalado y una reducción de los índices de competitividad en el mercado (Silva, 2018). Llegados a ese punto, las organizaciones comienzan a plantearse la necesidad de abordar uno de los mayores retos tecnológicos con los que se pueden encontrar: la adaptación y migración de su código legado (Book, 2013).

El proceso de migración o modernización de una aplicación supone trasladarla a un nuevo entorno tecnológico, manteniendo su funcionalidad y causando la menor disrupción posible en el entorno original (Bisbal, 1999). Pero hablar de entorno tecnológico diferente no implica necesariamente, en este caso, abandonar el Mainframe. Aunque existe cierta tendencia a pensar que estos sistemas están abocados a su pronta desaparición, muchas de las migraciones de las aplicaciones desarrolladas en ellos tienen como destino nuevamente el Mainframe. En estos casos los motivos para adentrarse en un proceso de cambio podrían ser:

- Falta de soporte técnico de alguno de los proveedores.
- Integración de aplicaciones en una nueva arquitectura Mainframe.
- Cambio en el lenguaje de programación.
- Modernización de la interfaz de usuario (*Screen Scraping*[3]).

Con independencia de cuál sea el entorno destino existen una serie de características comunes a cualquier proyecto de migración dentro del ámbito Mainframe:

- Proyectos de grandes dimensiones (Bennett, 1995).

[2] "Nadie se da cuenta de los chirriantes sistemas de software que manejan el mundo, hasta que fallan" (Traducido por Google Translator).

[3] Modernización de la User Interface para mejorar la usabilidad utilizando un middleware (Screen Scraping Tool) que se encarga de la interoperación (G. Salvatierra, 2013).

- Escasa documentación o documentación desactualizada (Bisbal, 1999), (De Lucia, 2008), que hace necesario estudiar el código para conocer la funcionalidad (Bisbal, 1999), (De Lucia, 2008).
- Se ignoran los requisitos no funcionales, como por ejemplo la mantenibilidad de la aplicación destino (Zou, 2003) o ser acorde con los requisitos de una nueva arquitectura (Hasselbring, Fuhr and Riediger,2011).
- Debe implementarse toda la funcionalidad (Bennett, 1995).
- Es difícil desacoplar funcionalidades de la aplicación origen para ser sustituidas progresivamente por el nuevo desarrollo (De Lucia, 2008).
- Evolución de la aplicación origen durante la migración (Bennett, 1995): integración de mejoras en la aplicación migrada al mismo tiempo que se desarrolla.
- Convivencia con otras aplicaciones que pueden ser, a su vez, sistemas legados (Charette, 2020):
    - Posible convivencia en el entorno de producción de la aplicación antigua con otras ya adaptadas a los requerimientos técnicos de la migrada.
    - Necesidad de ampliar los casos de pruebas para contemplar todos los escenarios posibles debido a la situación anterior.
- El impacto en las otras aplicaciones con las que se relaciona debe ser el menor posible (Book, 2013).
- La migración afecta tanto al software como a los datos (Bisbal, 1999). Con respecto a estos últimos, hay que minimizar el coste y el esfuerzo dedicado a su transformación, manteniendo la consistencia entre ellos y sin perderlos durante el proceso (Martens, 2018)
- También pueden aparecer dificultades para entender el funcionamiento del sistema, carecer de componentes e interfaces bien definidas o mantener dependencias de tecnologías o arquitecturas obsoletas (Fuentes-Fernández et al., 2012)

Ya sea por la aparición de alguno de los problemas aquí citados o por una combinación de todos ellos, los estudios sobre la relación éxito-fracaso de los

proyectos de modernización de aplicaciones no arrojan resultados muy halagüeños. Estos malos resultados tienen una repercusión directa en el aspecto económico llegando a provocar pérdidas millonarias. La revista Spectrum del IEEE mencionaba algunos ejemplos en un artículo sobre sistemas legados que publicó a raíz de la situación generada en los sistemas IT gubernamentales de EEUU por motivo de la pandemia de COVID-19. Por citar algunos de ellos:

- Departamento de Defensa de EEUU[4]. Tras 12 años de trabajo y más de un millón de dólares gastados, se canceló el proyecto de unificación de los 90 sistemas IT que tenían para la gestión de nóminas y personal.
- Departamento de Comunidades y Gobiernos Locales del Reino Unido[5]. El proyecto de reducción del número de centros de control del Departamento de Bomberos se canceló después de 6 años de trabajo y una inversión en él de 500 millones de libras.
- Ministerio de Servicios Sociales y Comunitarios de Canadá. Implantó un nuevo Sistema de Gestión de la Seguridad Social 18 meses más tarde de la fecha estimada y 40 millones de dólares por encima de lo presupuestado. Solucionar todos los problemas posimplantación supuso una cantidad de 52 millones de dólares que hubo que sumar a los 242 en los que estaba valorado el nuevo sistema.
- Lidl, Alemania. Intentó durante tres años hacer funcionar su nuevo sistema de gestión de mercancía valorado en 550 millones de euros, pero transcurrido ese tiempo y sin haber logrado que el nuevo sistema funcionase correctamente, tuvo que retomar el antiguo.

A la luz de estos resultados parece más que evidente la necesidad de estudiar cuáles pueden ser los problemas que se encuentran detrás del fracaso de estos proyectos o de otros de características similares. Standish Group publica periódicamente el informe CHAOS (Comprehensive Human Appraisal for Originating Software), con el que da una visión sobre el fracaso o éxito de los proyectos a lo largo de un determinado período de tiempo. Para ello utiliza una escala de tres valores: Éxito, Con dificultades y Fallido. Para la consideración de proyecto exitoso (primer grupo), Standish Group cambia su definición tradicional

[4] U.S. Department of Defense
[5] U.K. Department for Communities and Local Government

de “OnBudget, OnTime, OnTarget” (dentro del presupuesto, en la fecha estimada y con el alcance acordado) sustituyendo el concepto de “OnTarget” por la expresión “con un resultado satisfactorio”. Es decir, que un proyecto pasa a considerarse exitoso si, cumpliendo con los requisitos de presupuesto y fecha de entrega, el cliente queda satisfecho con el resultado obtenido y resta importancia a la cobertura del alcance que se haya logrado. En el segundo nivel (Con dificultades) se contabilizan los proyectos que han finalizado por encima del presupuesto, sobrepasando la fecha estimada de entrega o sin la completa satisfacción del cliente. El tercer grupo (Fallido) está reservado para aquellos proyectos que se abandonan o cancelan en algún momento, suponiendo una pérdida total de la inversión.

Al estudiar la evolución en los resultados cuando se confrontan los datos obtenidos en los informes de los años 2009 y 2015 (Figura 1) se observa claramente el efecto de este cambio de criterio. El porcentaje de proyectos fallidos anteriores al año 2010 es aproximadamente del 50%, mientras que a partir de esa fecha ese valor desciende a valores cercanos al 20%. El de proyectos exitosos aumenta, aunque la diferencia es menor (del 31% de media al 38%, aproximadamente) y donde sí se detecta una diferencia notable es en el conjunto de proyectos que finalizan con algún tipo de dificultad, que pasa de una media aproximada del 20% a otra del 43%. Una posible interpretación de estos resultados sería que, con el cambio de criterio, proyectos que antes veían prácticamente imposible cumplir con sus objetivos y se acababan abandonando, ahora continúan adelante y se finalizan, aunque su resultado no sea 100% exitoso. Otros, que antes no cubrían todos objetivos y entraban dentro del grupo de finalizados con dificultad, ahora engrosan la cifra de los que sí cumplen con todos ellos.

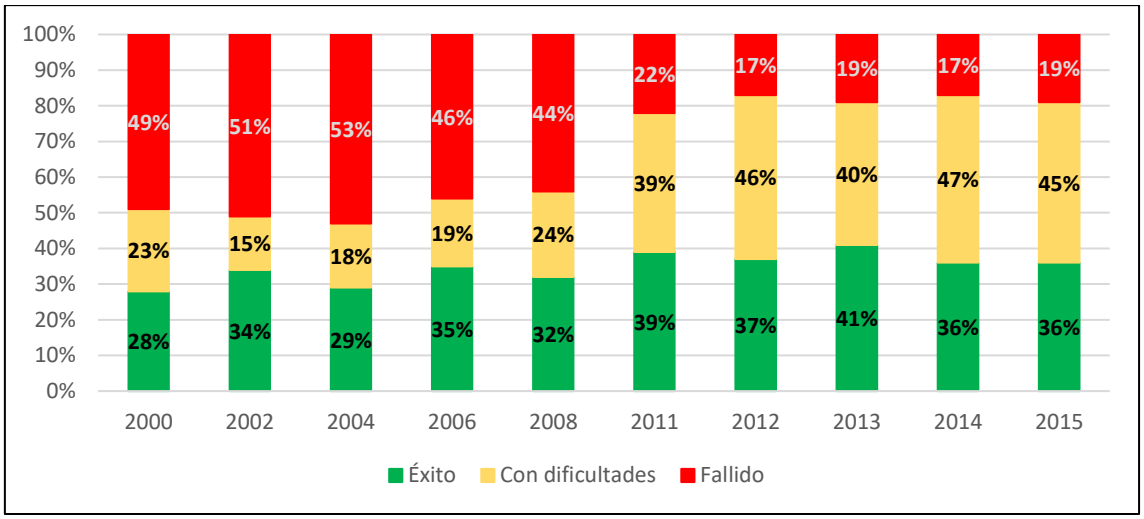


Figura 1 – Evaluación CHAOS por evolución anual.
Elaboración propia. Fuente: (CHAOS, 2015; CHAOS 2009)

El último de los informes de Standish Group puesto a disposición del público general corresponde al intervalo de años fiscales 2011-2015 y en él se observan los resultados que se obtienen a continuación.

El primero de los parámetros tenidos en cuenta en el estudio de los proyectos es su **tamaño**. En la Figura 2 puede observarse que el porcentaje de abandono es mayor en los de grandes dimensiones: 43% frente al 7% de los proyectos pequeños. Por lo general las aplicaciones Mainframe implementan productos de negocio funcionalmente complejos y eso las hace ser bastante voluminosas. Viendo que solo el 6% de los proyectos muy grandes cumplen con la definición de éxito y que en los pequeños el porcentaje se eleva hasta el 61%, el tamaño podría considerarse uno de los factores clave en el éxito de los proyectos.

Otro parámetro que destacar dentro del estudio de Standish Group es el tipo de **metodología** aplicado en la gestión del proyecto. Los datos que ofrece muestran las diferencias entre aquellos que siguieron el modelo tradicional en Cascada (Waterfall) y los que se desarrollaron de acuerdo con un paradigma ágil (sin especificar cuál de ellos en concreto). A tenor de los resultados (Figura 3), parece más beneficioso aplicar prácticas ágiles que seguir el modelo tradicional, ya que en el cómputo total de proyectos solo se abandonan un 9% de los clasificados como ágiles, mientras que en los tradicionales esta cifra sube hasta el 29%. También hay una gran diferencia si se comparan los proyectos considerados

totalmente exitosos, esto es, cumpliendo con todos los parámetros de la definición de éxito (39%-11%), pero los porcentajes quedan bastante igualados cuando lo que se coteja es la cantidad de proyectos que han finalizado con un éxito parcial o, lo que es lo mismo, con dificultades (52%-60%).

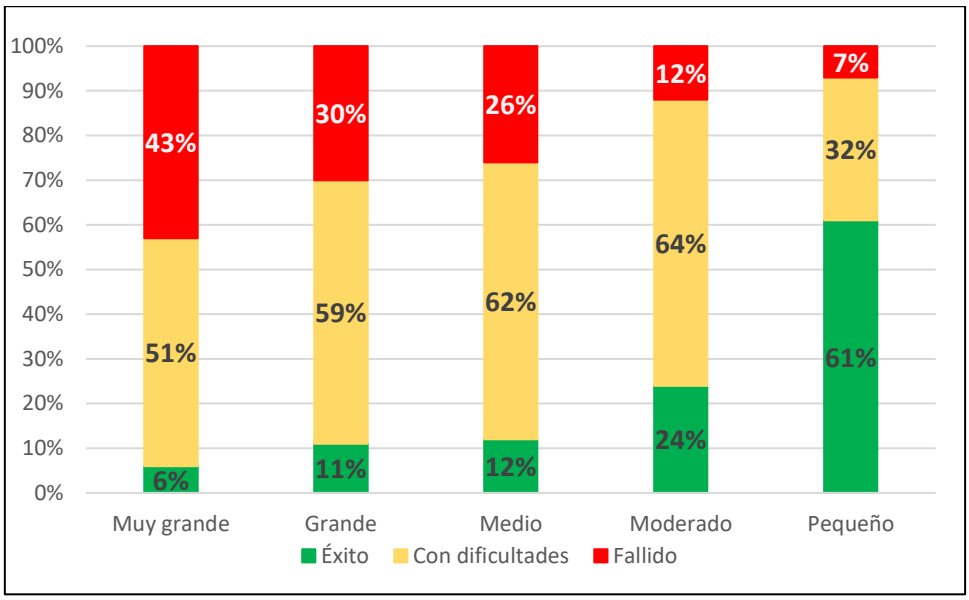


Figura 2 – Evaluación CHAOS por tamaño del proyecto.
Elaboración propia. Fuente: (CHAOS, 2015)

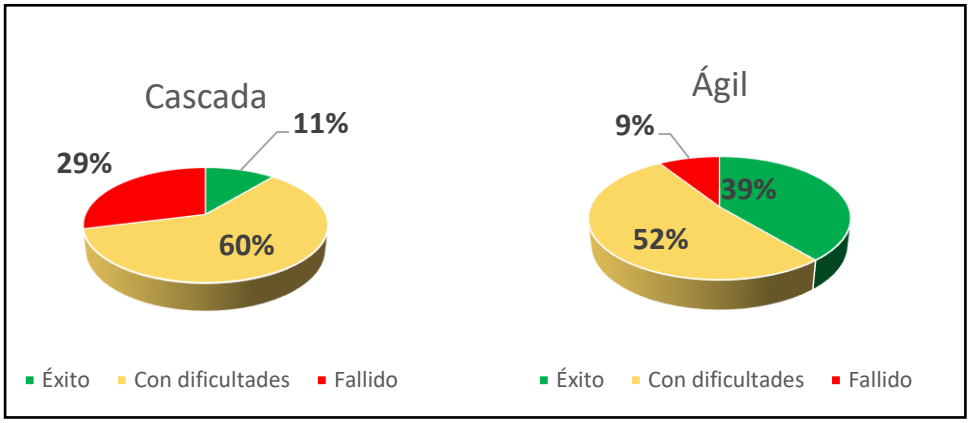


Figura 3 – Evaluación CHAOS por metodología aplicada.
Elaboración propia. Fuente: (CHAOS, 2015)

Si se añade a la comparativa de la metodología utilizada el tamaño del proyecto (Figura 4), se observa que el porcentaje de abandono en proyectos ágiles grandes (23%) es casi igual al de proyectos tradicionales medianos (25%), y el

de los ágiles medianos idéntico al de tradicionales pequeños (11% en ambos casos), lo que podría indicar que la utilización de técnicas ágiles palía en parte el efecto negativo que aporta el tamaño del proyecto. Sin embargo, el gráfico muestra también que, a menor tamaño, mayor porcentaje de éxito y menor de abandono, tanto en proyectos ágiles como en aquéllos otros en los que se haya optado por una gestión tradicional. Además, que la diferencia entre los porcentajes de cada sección (Éxito, Con dificultades, Fallido) se vea disminuida a medida que se reduce el tamaño del proyecto, hace pensar que tal vez no sea la metodología en sí la causante del fracaso o del éxito de los proyectos, sino que sea (nuevamente) su tamaño el factor más determinante para ello.

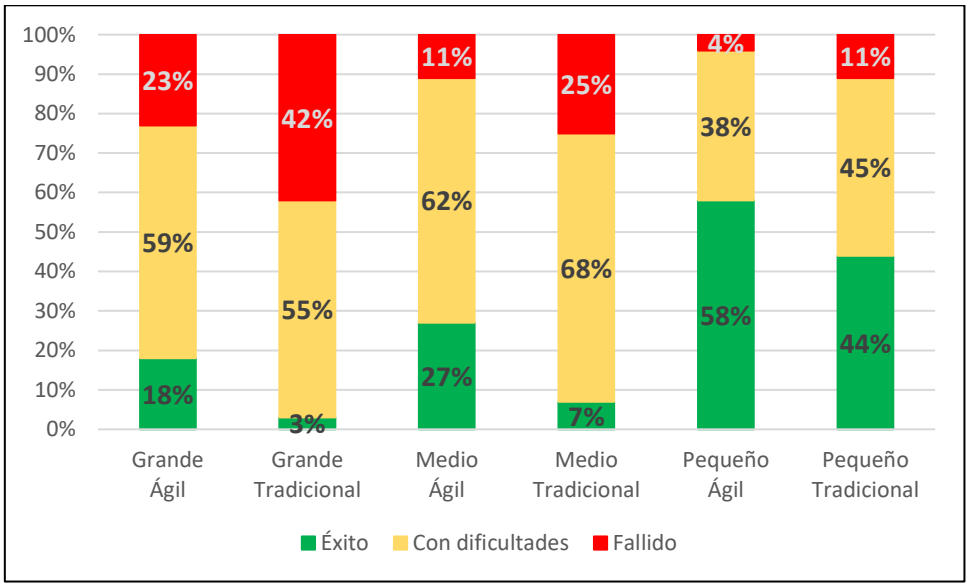


Figura 4 – Evaluación CHAOS por metodología y tamaño del proyecto.
Elaboración propia. Fuente: (CHAOS, 2015)

La **complejidad** (Figura 5) es otro factor importante que debe ser tenido en cuenta cuando se evalúan las posibilidades de éxito o fracaso de un proyecto. Entre los complejos y muy complejos, y entre los sencillos y muy sencillos no se aprecia mucha diferencia en los porcentajes de cualquiera de las bandas evaluadas (Éxito, Con dificultades, Abandonado), pero el salto de los dos primeros grupos a los de tamaño medio sí marca una diferencia apreciable en las bandas extremas, pasando de 28%-26% a 18% el porcentaje de abandono, y de 15%-18% a 28% el de éxito. El porcentaje de proyectos finalizados con dificultades observa su mayor escalón entre los proyectos medianos y los

sencillos, paso en el que disminuye en un 5%, pero en general presenta una tendencia bastante estable.

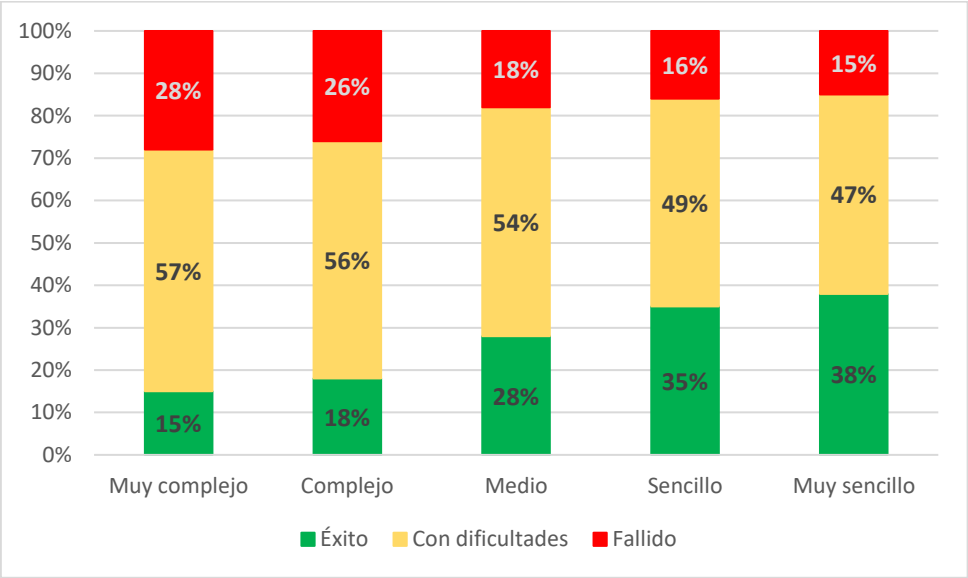


Figura 5 – Evaluación CHAOS por complejidad del proyecto.
Elaboración propia. Fuente: (CHAOS, 2015)

Como ya se ha comentado anteriormente, las aplicaciones del entorno Mainframe implementan productos de negocio funcionalmente complejos y cualquier proyecto que tenga alguna de estas aplicaciones como protagonista presentará un alto grado de dificultad funcional.

Otra perspectiva desde la que se presentan resultados en el informe CHAOS es la del **tipo de proyecto**. Una de las categorías analizadas es “Modernización”, que a primera vista podría interpretarse como la correspondiente a proyectos de migración. Sin embargo, atendiendo a las descripciones que se proponen en el propio informe, cualquiera de las demás categorías podría ser también interpretable como proyecto de migración o transformación de aplicaciones. Por ejemplo, los de “Desarrollo desde cero” podrían equivaler a un proyecto de migración en el que se reconstruye todo. Los que utilizan componentes externos o adquiridos, ya sea modificándolos o no, equivaldrían a una migración en la que se reutilizarían componentes del sistema legado. Siendo así, las conclusiones que se pueden extraer de la comparativa ofrecida en ese capítulo están más relacionadas con la técnica de migración aplicada que con las características de las aplicaciones en sí. Los casos de adquisición de la aplicación, con o sin

modificación de la misma, corresponderían a la no migración de la aplicación legada. En ellos, los porcentajes de abandono y de dificultad para la finalización del proyecto pondrían de manifiesto los problemas asociados al mantenimiento de este tipo de aplicaciones: pueden ser fuente de problemas en su ejecución diaria (en color amarillo) y, en situaciones más críticas, dejar de dar el servicio que se esperaba de ellas (en color rojo). Los resultados se recogen en la Figura 6. Sin tener en cuenta el 9% de los proyectos de modernización y el 25% de "Otros" por desconocer las condiciones de trabajo bajo las que se desarrollan, la media del porcentaje de abandono ronda el 18%. Como dato curioso se podría mencionar que, en este caso, el porcentaje de éxito es superior para los proyectos que se desarrollan aplicando métodos tradicionales, lo que a priori estaría en contradicción con la conclusión obtenida en el análisis de la metodología aplicada en la gestión del proyecto (segundo de los parámetros aquí expuestos), cuando se comparaba el sistema Waterfall con las técnicas ágiles. La conclusión que podría obtenerse de esta aparente contradicción es que son otros factores y no la metodología en sí los más determinantes en el éxito o fracaso de los proyectos, como ya se ha argumentado anteriormente.

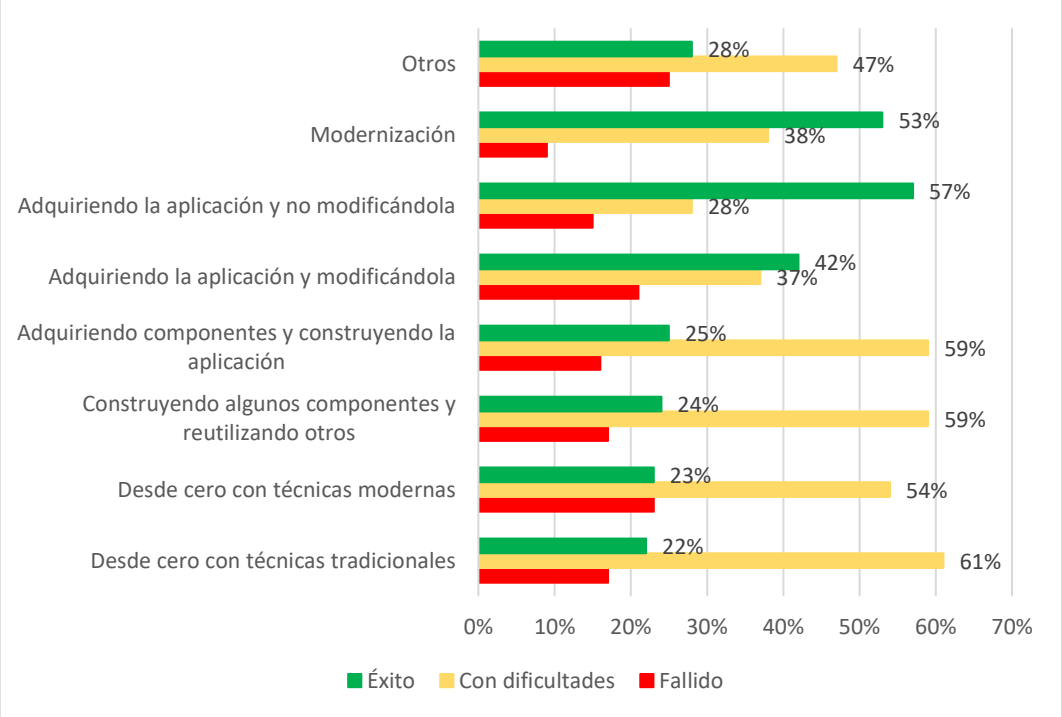


Figura 6 – Evaluación CHAOS por tipo de proyecto.

Elaboración propia. Fuente: (CHAOS, 2015)

Los demás puntos de vista desde los que se analiza el resultado de los proyectos en el informe CHAOS no se consideran relevantes para el objetivo que se busca con este TFM. Son, por ejemplo, los resultados por **sector** (banca, finanzas, gubernamentales, salud, etc.), **zona geográfica** (Asia, Europa, Norte América, etc.) o la **capacitación** de los participantes en el proyecto. El entorno Mainframe está presente en todos los sectores que se mencionan y en todos los países. En cuanto al personal, no se aportan datos que permitan comparar resultados entre los que participan en proyectos de migración, de desarrollo o por entorno.

En referencia a los proyectos de migración de sistemas legados Mainframe, se da la circunstancia de que en ellos suelen prevalecer las metodologías tradicionales y acostumbran también a ser los de mayores dimensiones. El tamaño y la complejidad que presentan les hace tener un riesgo de fracaso especialmente alto, por lo que para ellos es esencial contar con una metodología bien definida y fácil de implementar para la finalización exitosa del proyecto (Bisbal, 1999). Sin una metodología, o sin la metodología adecuada, muy probablemente habrá elementos que se pasen por alto, queden sin implementar o no se integren correctamente con el resto del sistema (Wong, Lee and Tshai, 2013).

2.1. Diseño metodológico

Para llevar a cabo la investigación propuesta se utilizan métodos cualitativos, principalmente de análisis de contenidos, considerando cono unidades de análisis los modelos de gestión de proyectos y los proyectos de migración de aplicaciones Mainframe del ámbito bancario.

Modelos y buenas prácticas de gestión de proyectos y migración de aplicaciones

Esta unidad de análisis la conforman los distintos modelos de madurez, estándares, metodologías y buenas prácticas contemplados dentro de la investigación. Se estudian manuales y artículos en los que se describen las principales características de cada uno de ellos, así como sus ventajas e inconvenientes. El material de estudio se recopila principalmente de bases de datos de conocimiento (ICYT, IEEE Xplore, etc.) y de los manuales y páginas web de las entidades propietarias de los estándares, metodologías y buenas prácticas analizados.

Proyectos de migración

Unidad de análisis formada por proyectos de migración conocidos por aparecer en artículos consultados en las bases de datos de conocimiento o por pertenecer al ámbito de trabajo real en entidades bancarias españolas. Dentro de este segundo grupo no se han tenido en cuenta únicamente aquéllos en los que se ha participado de manera directa desempeñando un rol principal (programador, analista o jefe de proyecto), sino también aquellos otros con los que se ha tenido relación por desempeñar en ese momento tareas de soporte técnico y funcional de la infraestructura sobre la que se construían las nuevas aplicaciones.

3. ESTADO DE LA CUESTIÓN

3. Estado de la cuestión

La migración de aplicaciones es una actividad que viene desarrollándose desde hace décadas y que consiste en obtener un sistema tecnológicamente nuevo manteniendo la funcionalidad del original. Como ya se ha visto anteriormente y con independencia de cuál sea el entorno destino, este tipo de proyectos presentan una serie de peculiaridades que no tienen los dedicados exclusivamente a la construcción de nuevas aplicaciones o al mantenimiento de aquellas que ya existen. Los modelos de referencia para la gestión de proyectos no suelen ofrecer propuestas específicas para aquellos que comparten características de desarrollo y de mantenimiento de software, lo que significa que ninguno de ellos puede garantizar por sí solo el éxito de una migración. En la práctica, esto trae como consecuencia que en las migraciones de software no sea habitual seguir ningún modelo metodológico o de buenas prácticas y se deje que el éxito del proyecto dependa, de manera casi absoluta, de las dotes organizativas del jefe de proyecto y de la experiencia previa de todos los componentes del equipo.

El proceso de generación de software debe observarse desde una doble perspectiva. De un lado está el plano de la producción o de la construcción de la aplicación en sí, y por otro el plano de la gestión o de las tareas dedicadas al establecimiento y organización de las distintas fases del proyecto y al correcto funcionamiento del equipo (MOMOCS-D32, 2007). Investigando la literatura dedicada a la migración de aplicativos, se encuentran propuestas para la gestión de este tipo de proyectos, pero la mayoría de ellas se centran más en el aspecto técnico de la conversión que en la propia gestión de los procesos implicados o en el establecimiento de buenas prácticas. A la hora de idear y proponer buenas prácticas o una nueva metodología resulta conveniente aplicar también esta “bifocalidad” y tener en cuenta tanto las estrategias de migración como las metodologías de gestión y organización de proyectos. Entre estas últimas estarían aquellas que se centran en la gestión de procesos y servicios, como ITIL, y las encaminadas a dirigir el desarrollo del software (Cascada, SCRUM).

Cuando en los capítulos siguientes se describen algunas de las técnicas de migración aparecidas a lo largo del tiempo, no se hace mención explícita a los entornos tecnológicos de origen y destino. El estudio se centra en conocer las

etapas o fases en las que queda organizado el trabajo y no en los detalles técnicos de una migración particular.

3.1. Metodologías de migración de aplicativos

Son varias las propuestas que han aparecido a lo largo de los años para abordar la problemática de la migración de aplicaciones. Aunque cada método hace una propuesta particular de cómo estructurar las etapas en las que subdivide el proceso completo, el esquema general de las fases que se deben completar es bastante similar en todos ellos (Bisbal, 1999):

1. Justificación de la migración
2. Estudio de la aplicación legada
3. Desarrollo de la aplicación destino
4. Pruebas
5. Puesta en producción

Las diferencias entre ellas están tanto en la estructura particular de fases que proponen, como en la técnica concreta de migración de código que se aplica (desarrollo de código nuevo, encapsulamiento de módulos o modificación del existente).

Desde el punto de vista estratégico existen dos grandes alternativas (Brodie and Stonebraker, 1993):

- **Cold turkey**, que supone convertir el sistema de una sola vez reescribiendo todo el código desde el principio.
- **Chicken little**, que contempla un proceso de conversión incremental e iterativo

La primera de estas dos opciones presenta multitud de riesgos debidos a que supone trabajar con equipos grandes y un gran volumen de software. Además, la alta disponibilidad que se requiere a las aplicaciones hace que deban estar operativas prácticamente el 100% del tiempo, dificultando con ello el proceso de entrada en producción. La segunda opción divide el trabajo en unidades más pequeñas e introduce el modelo de desarrollo iterativo, lo que hace más controlable el proceso de migración, no solo porque el equipo de trabajo y el volumen de software sean más reducidos, sino también porque los problemas que inevitablemente van a aparecer lo harán de manera escalonada.

3.1.1. Método de migración directa de sistemas divisibles[6]

Método aplicable a sistemas divisibles y que propone migrar la base de datos en un único paso “Cold turkey” y a continuación migrar la aplicación con estrategia “Chicken little”. Primero se adecúa la aplicación legada para que acceda a la nueva base de datos mediante un adaptador directo (*forward database gateway*) y después se aborda la conversión iterativa del código (Brodie and Stonebraker, 1993). Consta de los siguientes pasos:

- **F1** Instalación iterativa del entorno destino.
- **F2** Análisis del sistema legado.
- **F3** Descomposición del sistema legado.
- **F4** Diseño de las aplicaciones destino y sus interfaces.
- **F5** Diseño de la base de datos destino.
- **F6** Diseño e instalación del adaptador directo de la base de datos.
- **F7** Migración de la base de datos legada.
- **F8** Migración iterativa de las aplicaciones legadas.
- **F9** Migración iterativa de las interfaces legadas.
- **F10** Activar iterativamente el sistema destino.

Existen variantes de este método aplicables a sistemas semi-divisibles[7] o no divisibles[8]. En el primer caso, la dificultad añadida reside en los adaptadores de aplicación (*application gateway*) que deben construirse para permitir las llamadas entre la aplicación legada y la destino, y viceversa. Estos adaptadores de aplicación deben coordinarse con los adaptadores de base de datos (Brodie and Stonebraker, 1993). El modelo consta de los siguientes pasos:

- **S1** Instalación iterativa del entorno destino.
- **S2** Análisis iterativo del sistema legado.
- **S3** Descomposición iterativa del sistema legado.
- **S4** Diseño iterativo de las interfaces destino.
- **S5** Diseño iterativo de las aplicaciones destino.
- **S6** Diseño de la base de datos destino.
- **S7** Diseño e instalación del adaptador de aplicaciones.
- **S8** Migración de la base de datos.

[6] Forward Migration Method For Decomposable Legacy Iss (Brodie y Stonebraker, 1995).
[7] Migration Method For Semi-decomposable Legacy ISs (Brodie y Stonebraker, 1995).
[8] Migration Method For Non-decomposable Legacy ISs (Brodie y Stonebraker, 1995).

- **S9** Migración iterativa de las aplicaciones legadas.
- **S10** Migración iterativa de las interfaces legadas.
- **S11** Activar iterativamente el sistema destino (del legado).

En el caso de sistemas no divisibles se construye un adaptador del sistema que encapsula la totalidad de la aplicación legada (Brodie and Stonebraker, 1993). El modelo consta de los siguientes pasos:

- **N1** Instalación iterativa del entorno destino.
- **N2** Análisis iterativo del sistema legado.
- **N3** Descomposición iterativa de la estructura del sistema legado.
- **N4** Diseño iterativo de las interfaces destino.
- **N5** Diseño iterativo de las aplicaciones destino.
- **N6** Diseño iterativo de la base de datos destino.
- **N7** Creación e instalación iterativa del adaptador del sistema.
- **N8** Migración iterativa de la base de datos legada.
- **N9** Migración iterativa de las aplicaciones legadas.
- **N10** Migración iterativa de las interfaces legadas.
- **N11** Activar iterativamente el sistema destino (del legado).

3.1.2. Método de migración inversa de sistemas divisibles[9]

En este método también se propone la migración de la base de datos en un único paso “Cold turkey” para permitir migrar la aplicación con estrategia “Chicken little”. La diferencia con el método anterior estriba en la localización del paso de migración de la base de datos, que se sitúa después de la migración iterativa del código y no antes. Para ello se necesita un adaptador inverso para que la aplicación que se construye pueda acceder a la base de datos legada (*reversal database gateway*) (Brodie and Stonebraker, 1993). Consta de los siguientes pasos:

- **R1** Instalación iterativa del entorno destino.
- **R2** Análisis del sistema legado.
- **R3** Descomposición del sistema legado.
- **R4** Diseño de las aplicaciones destino y sus interfaces.

[9] Reverse Migration Method For Decomposable Legacy Iss (Brodie y Stonebraker, 1995).

- **R5** Diseño de la base de datos destino.
- **R6** Diseño e instalación del adaptador inverso de la base de datos.
- **R7** Migración iterativa de las aplicaciones legadas.
- **R8** Migración iterativa de las interfaces legadas.
- **R9** Migración de la base de datos legada.
- **R10** Activar iterativamente el sistema destino (del legado).

3.1.3. Método de migración general de sistemas divisibles[10]

Se trata de un método aplicable a cualquier sistema divisible. Contempla la creación de ambos tipos de adaptadores de base de datos (directo e inverso), lo que facilita que los pasos de migración puedan hacerse de manera iterativa y en paralelo, pero supone la utilización de un protocolo de commit en dos fases (two-phase-commit o 2PC) para asegurar la consistencia entre ambas bases de datos, la legada y la destino. Consta de los siguientes pasos (Brodie and Stonebraker, 1993):

- **D1** Instalación iterativa del entorno destino.
- **D2** Análisis del sistema legado.
- **D3** Descomposición del sistema legado.
- **D4** Diseño de las aplicaciones destino y sus interfaces.
- **D5** Diseño de la base de datos destino.
- **D6** Diseño e instalación del adaptador (gateway) de la base de datos.
- **D7** Migración iterativa de la base de datos legada.
- **D8** Migración iterativa de las aplicaciones legadas.
- **D9** Migración iterativa de las interfaces legadas.
- **D10** Activar iterativamente el sistema destino (del legado).

3.1.4. Metodología Butterfly[11]

La Metodología Butterfly (*Butterfly Methodology*) es una propuesta desarrollada por el Trinity College dentro del Proyecto Milestone en respuesta a los métodos descritos en los capítulos anteriores (Wu et al, 1997). Bajo su punto de vista, el uso de adaptadores (*gateways*) añade complejidad al proceso de migración y por ello ofrecen una solución en la que los sistemas legado y destino no conviven, haciendo innecesario el acceso cruzado entre ellos. En la Metodología

[10] General Migration Method For Decomposable Legacy ISs (Brodie y Stonebraker, 1995).
[11] Butterfly Methodology (Bing Wu et al, 1997).

Butterfly la construcción de la nueva aplicación y la migración de datos se desarrollan en diferentes fases, siendo la segunda de estas actividades su foco principal. Como características destacables pueden mencionarse:

- El gran porcentaje de tiempo dedicado a pruebas (puede llegar al 80%).
- Su flexibilidad (no impone el uso de ninguna herramienta concreta).
- Estimación precisa del tiempo que durará la migración.
- Mínima interacción entre sistemas legado y destino.
- Mínima interrupción del sistema legado (sigue activo hasta la entrada en producción del recién construido).
- Facilita la ejecución de tareas en paralelo.

En cuanto a la organización del trabajo, la metodología lo distribuye en las siguientes fases:

- **Fase 0** Preparación de la migración. Principalmente, requisitos de usuario y determinación del sistema destino.
- **Fase 1** Comprensión de la semántica del sistema legado y desarrollo del esquema (o los esquemas) del sistema destino.
- **Fase 2** Construcción de un Sample Datastore en el sistema destino, basado en la muestra de datos destino.
- **Fase 3** Migración incremental de los componentes del sistema legado a la arquitectura destino, exceptuando sus datos.
- **Fase 4** Migración gradual de los datos legados al sistema destino (actividad principal de la Metodología Butterfly) y formación de usuarios en el nuevo sistema.
- **Fase 5** Desconexión del sistema legado y arranque del sistema destino.

Cada una de estas fases lleva asociada una serie de actividades que desgranan el trabajo que se debe llevar a cabo dentro de ellas. Algunas son imprescindibles y otras opcionales.

3.1.5. Metodología XIRUP (eXrem end-User dRiven Process)

La Metodología XIRUP nace dentro del ámbito del proyecto europeo MOMOCS[12] (MOdel driven MOdernisation of Complex Systems) cuyo objetivo era el estudio de una metodología y sus herramientas relacionadas con el ánimo de conseguir

[12] https://cordis.europa.eu/project/id/034466/es (22-09-2020)

una rápida reingeniería de sistemas complejos (MOMOCS D31, 2007). Los principios en los que se basa son los siguientes:

- Procesos iterativos dirigidos por funcionalidades. Al transformar la aplicación por partes se minimiza el riesgo y se facilita la aceptación por parte del usuario.
- Ingeniería dirigida por modelos. Ofrece el nivel de abstracción recomendable para enfrentarse a sistemas complejos.

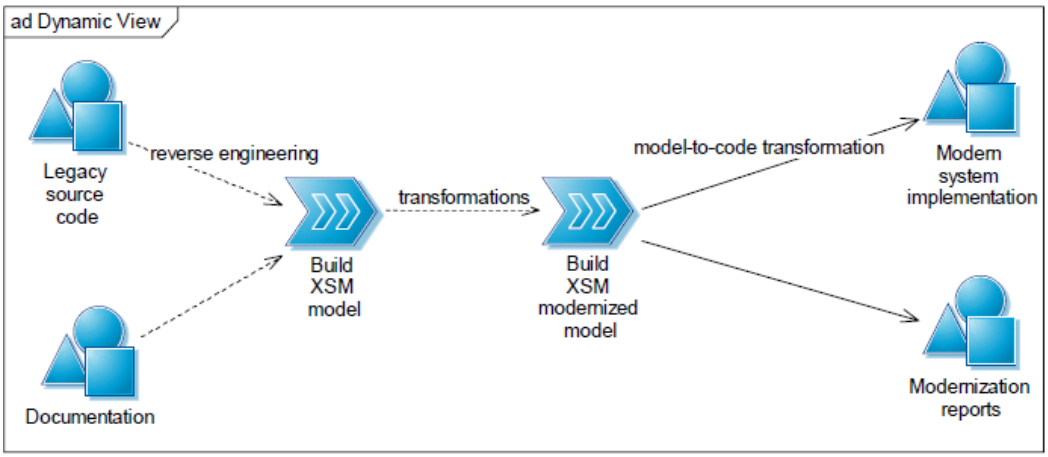


Figura 7 – Vista simplificada del trabajo con modelos en XIRUP.

Fuente: [21] (MOMOCS D31, 2007)

- Modernización basada en componentes. En ambas arquitecturas, legada y nueva. Permite encapsular el código legado.
- Alto grado de validación. Se realiza en cada una de las actividades ejecutadas, teniendo en cuenta requisitos funcionales y no funcionales.

El trabajo se distribuye en cuatro fases (Fuentes-Fernández et al., 2012):

- **Preliminar** Se decide, de acuerdo con una estimación de costes, si la migración se lleva a cabo o no (MOMOCS D31, 2007).
- **Comprensión** (del sistema y de los requisitos). Se recopila información referente al sistema legado, a la plataforma destino, a las transformaciones necesarias y a las restricciones que limitan la construcción del nuevo sistema.
- **Construcción** (del sistema nuevo). Se establecen las transformaciones entre los componentes del sistema legado y los del nuevo que se construye.

- **Migración** Despliegue en la nueva plataforma del código generado, migración de los datos y configuración del entorno destino para permitir la convivencia de elementos legados y nuevos.

3.1.6. FASMM (Fast and Accessible Software Migration Method)

La propuesta FASMM tiene su origen en un estudio que La Universidad Paris 1 Panthéon-Sorbonne hizo sobre un proyecto de migración desarrollado por una empresa francesa (Forite and Hug, 2014). Los principales problemas que se detectaron en este estudio fueron la falta de estandarización en los procesos de conversión de software y el no haber aprovechado el conocimiento adquirido a lo largo de la migración. Con el fin de proponer un modelo que diera solución a estos inconvenientes se establecieron los siguientes objetivos:

- Automatizar algunas de las fases de la migración.
- Asegurar que las funcionalidades migradas cumplían con sus objetivos técnicos y de negocio.
- Formalizar y capitalizar el conocimiento adquirido durante la migración.
- Definir un método sencillo que permitiera ser aplicado por desarrolladores no expertos en modelado o metamodelado.

La principal novedad con respecto a otros métodos radica en el tercero de estos objetivos, que queda implementado con la utilización de un diccionario de reglas de transformación que los propios desarrolladores van alimentando a lo largo del proyecto.

En lo que respecta a la organización del trabajo, en esta propuesta no hay una delimitación de fases propiamente dicha. En su lugar se definen cinco tareas objetivo que se irán repitiendo iterativamente hasta la completa migración del sistema:

- **Obtener un modelo** por ingeniería inversa, revisando el código y el modelo ya existente, e identificando el código reutilizable y el no aprovechable.
- **Migrar funcionalidades** manualmente mediante la aplicación de reglas de transformación, o de manera semiautomática con generación de código a partir de los modelos obtenidos.
- **Validar** lo migrado mediante pruebas técnicas y funcionales.

- **Descubrir reglas de transformación** a través de la experiencia, del análisis de modelos, de la identificación de problemas y de la revisión del código.
- **Enriquecer el diccionario** creando nuevas reglas, corrigiendo las ya existentes o reorganizando el propio diccionario.

3.1.7. ARTIST (Advanced software-based seRvice provisioning and migraTIon of legacy Software)

Entre los años 2012 y 2015 la Comisión Europea subvencionó el proyecto ARTIST[13]. En él participaron instituciones públicas y privadas de Alemania, Austria, Bélgica, España, Francia, Grecia, Italia y Turquía, y dio como resultado un acercamiento a la migración de software en el que, además de una fase de migración basada en técnicas MDE (Model Driven Engineering), se incluyen otras dos: premigración y postmigración (Menychtas et al., 2013) (Bergmayr et al., 2013).

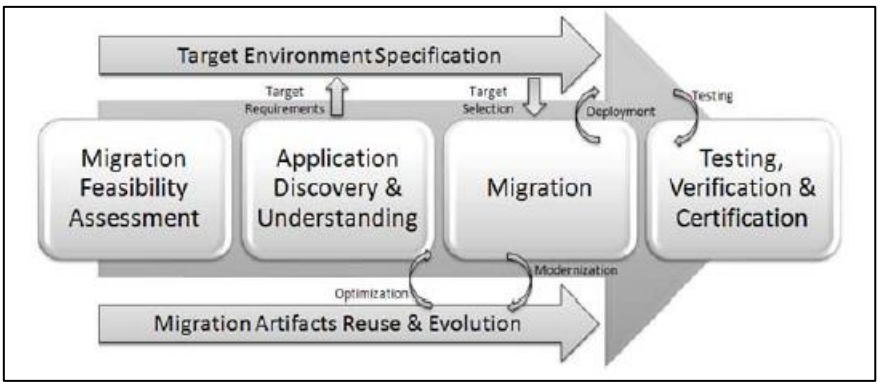


Figura 8 – Visión general de la metodología ARTIST.
Fuente: (Menychtas et al, 2013.)

- **Premigración** Se estudian las consecuencias técnicas y económicas de las distintas estrategias de migración que puedan ser aplicadas al proyecto. A partir de este estudio se definen los objetivos y se decide cómo se desarrollará la migración a lo largo de las siguientes fases (Bergmayr et al., 2013).
- **Migración** En esta fase se utiliza la ingeniería inversa (reverse engineering, RE) para el estudio del sistema legado y la ingeniería directa (forward engineering, FE) para la construcción del nuevo (Menychtas et

[13] https://cordis.europa.eu/project/id/317859/es (25-09-2020)

al., 2013). La aplicación de la ingeniería inversa permite obtener un modelo que recoge las características y los condicionantes sobre los que está construido el sistema legado (Legacy Platform-Specific Model, PSM). Con el fin de obtener un modelo que sea lo más reutilizable posible en los diferentes escenarios de migración que se puedan plantear, se aplica un proceso de abstracción al Legacy PSM que da como resultado otro modelo independiente de factores como el entorno de ejecución o la gestión de datos. Se logra así un modelo independiente de la plataforma (Platform-Independent Model, PIM), que debe ser adaptado a las características ofrecidas por el entorno destino para construir el software migrado (Bergmayr et al., 2013).

- **Postmigración** Se realizan pruebas de equivalencia para comprobar que el software tiene el comportamiento esperado y se evalúa la ejecución desde puntos de vista funcionales y no funcionales para certificar que se cumplen los objetivos de migración establecidos con anterioridad (Bergmayr et al., 2013). También en esta fase se despliegan los componentes en el entorno destino (Menychtas et al., 2013).

3.1.8. MADIISH (Metodología Ágil de Desarrollo de Software Incremental e Iterativa para la migración de Sistemas Heredados)

El Departamento de Sistemas y Computación del Instituto Tecnológico de Veracruz, tras haber estudiado metodologías de software tradicionales y ágiles, y metodologías de migración de sistemas heredados basados en el uso de herramientas de minería de datos KDD[14] (Knowledge Discovery in Databases), presentó, en 2018, la metodología MADIISH, caracterizada por ser incremental (la migración se lleva a cabo por subsistemas), iterativa (permite la modificación constante de un mismo subsistema) y ágil (disminuye considerablemente la generación de documentación requiriendo a cambio una participación constante del usuario) (Silva et al., 2018). En cuanto a las fases que la componen, son las que se recogen en el siguiente gráfico:

[14] Para las metodologías de tipo KDD el conocimiento de los sistemas heredados está implícito en los datos (Silva et al., 2018).

Figura 9 – Fases de la metodología MADIISH. Fuente: (Silva et al., 2018)

- **Fase C** Se define la estructura de sistemas y subsistemas y la interoperabilidad entre ellos. También se recogen las necesidades del cliente y se establece la estrategia a seguir.
- **Fase 1** Se decide qué subsistema migrar y se analiza en detalle para determinar la entrada y salida de datos, relación con otros subsistemas, nuevos requisitos de usuario, etc.
- **Fase 2** Se realiza el modelado de la base de datos correspondiente al subsistema que se migra, se desarrollan las interfaces de usuario y se desarrolla un prototipo para mostrar al usuario. Contempla volver a Fase 1.
- **Fase 3** Se establece un entorno de pruebas lo más parecido posible al de ejecución real en el que verificar si el comportamiento es el esperado. En caso de no ser así se regresa a la Fase 1.
- **Fase 4** Se despliega el subsistema migrado en el entorno de producción y se deshabilita el heredado. También se modifica la base de datos y entran en funcionamiento los módulos de interoperabilidad entre subsistemas.
- **Fase 5** Se elabora la documentación del subsistema migrado (procesos, código fuente, descripción gráfica, etc.). Contempla volver a Fase 1.

- **Fase T** Se revisa y completa la documentación global del sistema.

Al igual que otras metodologías que se han presentado en los capítulos anteriores, la propuesta MADIISH incluye el desarrollo incremental e iterativo y la existencia de módulos que facilitan la interoperabilidad entre el sistema nuevo y el migrado, pero en esta ocasión la iterabilidad no consiste solo en aplicar las mismas etapas en la migración de distintas funcionalidades o subsistemas, sino que menciona de manera explícita su aplicación sobre subsistemas ya migrados si estos no cumplen con los objetivos de desarrollo. También aporta como novedad la creación de un prototipo.

3.2. Estándares aplicables en entornos Mainframe

Con el fin de cubrir todos los aspectos de interés y obtener un punto de partida para la elaboración de un nuevo modelo de trabajo orientado al campo específico de aplicativos bancarios Mainframe, se exponen a continuación algunos de los estándares más comunes utilizados en este ámbito.

Según la definición de la Real Academia de la Lengua (RAE) un proceso puede definirse como el “conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial”. Los modelos de gestión no especifican cómo se deben implementar los procesos, sino que aportan una serie de buenas prácticas que ayudan a mejorarlos y que pueden servir como punto de partida para la elaboración de las que vayan a regir en el nuevo método de trabajo. El análisis de estos modelos ayudará a determinar las fases en las que quedará estructurado el nuevo modelo de trabajo. Al tratarse de un estudio “auxiliar” (no es el objetivo principal del TFM conocer en profundidad las metodologías y estándares que aquí se exponen) se hace una presentación general de las mismas y centrada en los aspectos que se aplicarán en el nuevo modelo de trabajo que se desarrolla en capítulos posteriores.

3.2.1. ITIL® (Information Technology Infrastructure Library)

ITIL se define como “una colección de publicaciones, o librería, que describen de manera sistemática un conjunto de “buenas prácticas” para la gestión de los servicios de Tecnología Informática (TI)”. Aporta la idea y la estructura que debe tener el plan que se ponga en marcha, pero no indica con qué herramientas hay que llevarlo a cabo (2011, ITIL v3).

ITIL v3 (2011) lo conforman 26 procesos repartidos en las cinco etapas que componen el **Ciclo de Vida del Servicio** (Figura 10): Estrategia del servicio, Diseño del servicio, Transición del servicio, Operación del servicio y Mejora continua del servicio.

Figura 10 – Ciclo de vida ITIL v3 (2011)

Fuente: https://wiki.en.it-processmaps.com/index.php/Main_Page

1. **Estrategia del servicio** (Service Strategy)

Tiene como objetivo introducir las TI en la estrategia empresarial alineando la tecnología con el negocio. Para ello, la organización debe conocer con qué recursos tecnológicos cuenta, hasta dónde puede llegar con ellos o si necesita invertir para mejorar su infraestructura TI antes de comenzar a gestionar su futuro.

2. **Diseño del servicio** (Service Designe)

Tiene como objetivo diseñar nuevos servicios, así como modificar y mejorar los ya existentes. Estas tareas de diseño y modificación deben estar alineadas con la estrategia establecida y, dado que las modificaciones de los servicios se deben a los requerimientos de los clientes, ésta a su vez debe adaptarse a los cambios que se vayan produciendo en los servicios.

3. **Transición del servicio** (Service Transition)

Tiene como objetivo construir y desplegar servicios asegurando que el cambio de los servicios y los procesos de gestión de servicios se llevan a cabo de manera coordinada.

4. **Operación del servicio** (Service Operation)

Tiene como objetivo asegurar que el servicio se presta de manera eficiente y eficaz. Comprende actividades de atención a las peticiones de los usuarios, resolución de fallos del sistema y arreglo de problemas, además de otras tareas más rutinarias.

5. **Mejora continua del servicio** (Continual Service Improvement)

Quinta etapa del ciclo de vida. Tiene como objetivo mejorar continuamente la eficiencia y eficacia de los servicios TI, para lo que se aplican métodos de gestión de la calidad que permiten aprender de los éxitos y los errores pasados. Sigue las líneas de la mejora continua adoptada en ISO 20000.

De estas cinco etapas será únicamente la de Transición del servicio a la que se haga referencia más adelante, concretamente los procesos de Gestión del Cambio y de Evaluación del Cambio:

- **Gestión del Cambio**

  Proceso que tiene como objetivo planificar, analizar y evaluar los cambios para que puedan efectuarse con la menor interrupción posible del servicio. Consta de los subprocesos:

  - Soporte a la gestión del cambio.
  - Evaluación de las propuestas de cambio.
  - Registro y revisión de RFC.
  - Evaluación e implementación de cambios de emergencia.
  - Evaluación del cambio por el CAB[15].
  - Planificación de cambios y autorización de implementación.
  - Autorización del despliegue del cambio.
  - Despliegue de cambios menores.
  - Revisión postimplementación y cierre del cambio.

[15] Change Advisory Board

- **Evaluación del Cambio**

  Proceso que tiene como objetivo estudiar los cambios que se quieran realizar antes de que pasen a la siguiente fase de su ciclo de vida.

  Consta de los subprocesos:

  - Evaluación del cambio antes de la planificación.
  - Evaluación del cambio antes de la implementación.
  - Evaluación del cambio antes del despliegue.
  - Evaluación del cambio después del despliegue.

3.2.2. Ciclo de vida tradicional: Modelo en Cascada

El modelo de desarrollo en cascada fue propuesto por primera vez en el año 1970 por Winston W. Royce (Royce, 1970). Aunque en ese mismo artículo mencionaba los inconvenientes que podría acarrear su aplicación estricta, fue extensamente adoptado e incluso hoy en día continúa aplicándose en algunos sectores del desarrollo de software. El modelo de desarrollo en cascada o "Waterfall" se basa en la secuenciación de las distintas fases de producción del software. En cada una de ellas se genera una documentación que sirve de entrada para la siguiente, es decir, que el trabajo que se lleva a cabo en cada fase se basa en el resultado de la anterior. Esto hace que no pueda abordarse una nueva etapa mientras no haya finalizado aquella que la precede (Gómez Palomo and Moraleda Gil, 2020).

Las fases en las que se divide el modelo en cascada son (Sommerville, 2005):

- Análisis y definición de requerimientos
- Diseño del software del sistema
- Implementación y pruebas de unidades
- Integración y pruebas de sistema
- Funcionamiento y mantenimiento

Se describen a continuación indicando también los documentos que se generan en cada una de ellas (Gómez Palomo and Moraleda Gil, 2020):

Análisis y definición de requerimientos

Los usuarios indican los requisitos y objetivos del sistema, cuya definición detallada conforma la especificación del sistema. Estas especificaciones quedan

recogidas en el Documento de requisitos de software (SRD[16], por sus siglas en inglés).

Diseño del software y del sistema

Los requisitos se subdividen en requerimientos hardware y software. Es la etapa donde se establece la arquitectura completa del sistema. En el diseño del software se identifican y describen las abstracciones fundamentales del sistema y sus relaciones. El documento donde queda recogida esta información (descripción global del sistema, función de cada una de las partes y la combinación de unas con otras) es el Documento de diseño de software (SDD[17], por sus siglas en inglés).

Implementación y pruebas de unidades

Se construyen los elementos diseñados en la etapa anterior y se prueban unitariamente para verificar que cada uno de ellos cumple con sus especificaciones. La documentación generada en esta etapa es el propio código fuente de los programas (comentado, para mayor claridad), sus ejecutables y los documentos de las pruebas realizadas.

Integración y prueba de sistema

Se prueban todos los elementos desarrollados en conjunto, como un sistema, para verificar que se cumplen los requerimientos de software definidos por el usuario.

Funcionamiento y mantenimiento

Se procede a la instalación y explotación del sistema. A partir de ese momento comienza la tarea de mantenimiento del software, en la que se corrigen errores no descubiertos en las etapas anteriores, se mejora la implementación del sistema y se modifican elementos según van apareciendo nuevos requisitos. Para dejar constancia escrita de todo ello se generan Documentos de cambio, que recogen el problema detectado, la solución adoptada y las modificaciones realizadas sobre el sistema para solventar el problema.

[16] Software Requirements Document.
[17] Software Design Document.

3.2.3. Metodología Ágil: Scrum

En el año 2001 se dieron a conocer los cuatro postulados en los que se basa el Manifiesto Ágil y que representan los valores en los que se fundamentan las metodologías ágiles. Estos cuatro principios básicos son:

- **Personas e interacciones sobre procesos e instrumentos.** Los procesos ayudan al trabajo, pero es necesario el talento y el conocimiento de las personas que los ejecutan para aportar creatividad e innovación.
- **Software que funciona sobre documentación completa.** La presentación de un prototipo o la ejecución de una parte del software ya terminada aporta más a la comunicación que la lectura de una documentación sobre requisitos excesivamente pesada. Hay que mantener la generación de documentos siempre que tengan como objetivo respaldar los hechos, registrar información histórica o cuestiones legales, facilitar la transferencia de conocimientos, etc. Es decir, siempre que aporte valor al producto.
- **Colaboración del cliente sobre negociación del contrato.** En un proyecto ágil el producto a generar se considera en evolución continua y la prioridad se centra en darle el mayor valor posible para el cliente a medida que se va produciendo esta evolución. Esto hace que se prefiera mantener una comunicación continua con el cliente frente a la elaboración de un documento de requisitos cerrado desde el inicio.
- **Respuesta al cambio sobre seguimiento de un plan.** Al trabajar con productos de requisitos inestables se valora más la capacidad de adaptación y respuesta que la de planificación y control que garantice el cumplimiento del plan inicial de trabajo.

Scrum es una de las metodologías nacidas al amparo del Manifiesto Ágil y que se caracteriza por:

- Desarrollo incremental. Gestión evolutiva en lugar de predictiva.
- Calidad del resultado basada en el conocimiento tácito de los componentes de equipos autoorganizados, en vez de en los procesos empleados.
- Fases del desarrollo solapadas en vez de secuenciales.

Los elementos principales sobre los que se construye esta metodología son el "Incremento" (cada parte del producto terminada, que funciona y es operativa) y el "Sprint" (periodo de tiempo dedicado a la construcción de este Incremento), y en cuanto a cómo queda organizada técnicamente aparecen los conceptos de Roles, Artefactos y Eventos (o Ceremonias):

Roles

El rol de una persona que participa en un proyecto Scrum determina su función dentro del mismo

- **Propietario del producto (Product owner).** Es la persona responsable de lograr el mayor valor del producto para los clientes, usuarios y resto de implicados.
- **Equipo de desarrollo.** Se trata de un equipo compuesto (preferiblemente) por entre tres y nueve personas, multifuncional, autogestionado y con responsabilidad compartida, que presenta espíritu de colaboración y que comparte el objetivo común de conseguir el mayor valor posible para el cliente.
- **Scrum Master.** Es la persona responsable de la correcta aplicación de las normas de Scrum y de facilitar el trabajo de todo el equipo, eliminando cualquier impedimento que pueda surgir a lo largo del proyecto.

Artefactos

Los artefactos son los elementos físicos que se generan en la aplicación de Scrum. Los tres principales son:

- **Pila del producto (Product backlog).** Recoge los requisitos del producto (funcionalidades, mejoras, tecnología y corrección de errores) en forma de historias de usuario y refleja qué se ha hecho y qué queda por hacer.
- **Pila del Sprint (Sprint backlog).** Recoge el conjunto de tareas que deben realizarse dentro del periodo de tiempo marcado por el Sprint para generar el incremento deseado, esto es, para construir las historias de usuario que se haya decidido incluir en él.
- **Incremento.** Cada parte del producto terminada, que funciona y es operativa, pero habría que añadir además que puede entregarse al

cliente. Los prototipos, módulos, submódulos, elementos sin probar o sin la documentación necesaria no forman parte del incremento.

- **Gráfico de avance del producto (burn-up).** Herramienta de planificación utilizada por el propietario del producto que, atendiendo a la velocidad del quipo, representa el avance previsible en la construcción del producto
- **Gráfico de avance del sprint (burn-down).** Herramienta de seguimiento que mide el trabajo pendiente de realizar y que permite al equipo de desarrollo controlar si el ritmo de avance del sprint se corresponde con el previsto.

Eventos (o ceremonias)

Los eventos que prevé Scrum son cinco.

- **Sprint.** Cada una de las iteraciones del desarrollo.
- **Planificación del Sprint.** Reunión que marca el inicio del sprint.
- **Sprint diario.** Reunión que el equipo mantiene diariamente durante un máximo de 15 minutos para sincronizar el trabajo y establecer el plan a seguir durante las siguientes 24 horas.
- **Revisión del Sprint.** Reunión que marca el final del sprint.
- **Retrospectiva.** Reunión que celebra el equipo de desarrollo después de la revisión del sprint y antes de la de planificación del siguiente con el objetivo de mejorar el marco de trabajo.

4. RESOLUCIÓN

4. Resolución

En los capítulos anteriores se ha visto cuál es el escenario en el que se desenvuelven los proyectos de migración Mainframe y las propuestas metodológicas que han ido apareciendo a lo largo de los años para afrontar la migración de sistemas de software. También se ha visto que, pese a la aplicación de estas metodologías, muchos de los proyectos no finalizan con el éxito esperado y se terminan fuera de plazo, por encima del presupuesto o, en el peor de los casos, se acaban abandonando. El modelo de trabajo que se propone pretende dar solución a los problemas que se mencionaban en el primer capítulo como principales inconvenientes de los proyectos de migración y que pueden resumirse en:

- Conocimiento insuficiente del entorno y tecnología destino.
- Determinación inexacta o irreal del alcance.
- Falta de modularidad en el diseño y desarrollo del nuevo sistema.
- Desconocimiento de la aplicación legada desde el punto de vista técnico.
- Problemas al compaginar el nuevo desarrollo con las modificaciones de la aplicación original que deben ser repercutidas en la nueva que se construye.

El objetivo de la propuesta es aumentar las posibilidades de éxito de los proyectos de migración (entendido el éxito como finalización en la fecha prevista, dentro del presupuesto y con satisfacción del cliente) complementando los aspectos que las metodologías estudiadas no tienen en cuenta o no especifican de una manera detallada. Así mismo, con el fin de facilitar a los jefes de proyecto la aplicación del nuevo modelo de trabajo, se propone el uso de diferentes herramientas de trabajo, ya sean estas de construcción propia, de código libre o comerciales.

4.1. Justificación del modelo de desarrollo propuesto

Tal y como quedaba reflejado en el informe CHAOS, uno de los factores más importantes que deben ser tenidos en cuenta a la hora de estimar las posibilidades de éxito de un proyecto es su **tamaño**, siendo ambos conceptos (tamaño y éxito) inversamente proporcionales. Es deseable por tanto que el proyecto (o unidad de gestión) sea lo más pequeño posible.

El siguiente factor que favorece una finalización correcta de los proyectos según este mismo informe es la aplicación de **metodologías ágiles**. Habitualmente en el entorno Mainframe la inercia lleva a optar en el desarrollo por soluciones más tradicionales, como el modelo en cascada. Un giro demasiado brusco hacia un enfoque totalmente ágil podría desestabilizar el desarrollo, puesto que el esfuerzo de adaptación a este nuevo paradigma acapararía gran parte de la atención y haría peligrar el verdadero objetivo del proyecto. La alternativa pasaría por adoptar una solución de compromiso, es decir, un método híbrido que, manteniendo el esquema tradicional de fases del modelo en cascada, incorpore elementos característicos de las metodologías ágiles.

En lo que se refiere a la **complejidad**, también es deseable que sea lo más baja posible, en principio algo complicado de conseguir debido a la propia naturaleza de las aplicaciones Mainframe, pero que no se desecha como objetivo a cumplir. Para dar solución a los problemas que se han planteado hasta a hora se recurre a las dos propuestas que existen desde el punto de vista estratégico: los modelos Cold turkey y Chicken Little. La ventaja de Chicken Little es la descomposición del trabajo en unidades más pequeñas que se van construyendo de manera iterativa (disminuye el tamaño de la unidad de gestión e introduce características de las metodologías ágiles), pero tiene el inconveniente de que precisa ir haciendo constantes adaptaciones a medida que esas unidades van entrando en producción, lo que provoca un aumento de la complejidad y la dilatación del proyecto en el tiempo (condición que debe ser evitada todo lo posible). El modelo Cold turkey acumula gran cantidad de riesgos, pero la implantación del nuevo sistema se realiza de una sola vez y evita la constante modificación del software una vez terminado. La solución que se propone y que intenta aprovechar las ventajas de ambos métodos es un desarrollo iterativo con implantación global. Esto es, todas las partes que se desarrollan por separado entran en producción al mismo tiempo.

Fuera ya del estudio CHAOS y atendiendo a las dificultades propias de los proyectos de migración, uno de los principales retos es ir adaptando la nueva aplicación a la **evolución y** los **cambios** sufridos por la aplicación origen durante el tiempo que dura el desarrollo. Todas esas modificaciones deben integrarse en la nueva aplicación previamente a la entrada en producción, ya que el cambio de una aplicación a otra no puede dar lugar a una pérdida de funcionalidad para el

usuario. Como estos cambios deben hacerse de manera ordenada e interfiriendo lo menos posible en el avance del desarrollo, se recurre al proceso de Gestión de Cambios de ITIL para diseñar una fase dedicada a esta tarea.

En resumen, el modelo de desarrollo propuesto se define como predictivo en cascada con desarrollo iterativo, incremental y adaptativo:

- **Predictivo:** las fechas de inicio y fin se establecen al comienzo del proyecto y deben respetarse.
- **En cascada**: el desarrollo del software se realiza siguiendo las etapas clásicas de esta metodología (diseño, codificación, pruebas).
- **Iterativo:** la aplicación se descompone en unidades más pequeñas cuya construcción se lleva a cabo en ciclos separados y repetitivos.
- **Incremental:** la aplicación se construye desarrollando subsistemas.
- **Adaptativo:** la funcionalidad del software se adapta a los cambios que se producen en la aplicación original al mismo tiempo que se construye la nueva.

4.2. Herramientas de apoyo

Para facilitar la aplicación del modelo de trabajo que se propone es importante apoyarse en el uso de herramientas, ya sean estas de construcción propia y particularizadas para el proyecto en cuestión, de código abierto o de uso comercial.

- **Formularios ad hoc de tipo check list:** en las fases de recogida de información o de preparación de entornos facilitan y agilizan la elaboración de listas de control. Con ellos también es más sencillo conseguir una estandarización en el aspecto de los documentos de trabajo y en la presentación de resultados, ya que es la propia herramienta la que se encarga de estas tareas.
- **Herramienta ad hoc para valoraciones:** aportan estandarización en la fase de valoración del esfuerzo. Al ser de creación propia pueden aplicarse baremos particularizados y adaptados a las características individuales de cada proyecto. Diseñada y parametrizada por analistas con más experiencia, son de gran ayuda para otros más noveles en la tarea de estimación de esfuerzos.

- **JIRA**[18]**:** herramienta online de la empresa Altassian para el seguimiento de incidencias y la definición de requisitos en proyectos ágiles. Permite establecer flujos de trabajo que facilitan la estandarización de la gestión de incidencias o la implementación de mejoras. Estos flujos de trabajo se pueden particularizar para adaptarlos a la estructura o dinámica de trabajo de la instalación donde se desarrolla el proyecto. Es una herramienta de gran utilidad en las fases de construcción de software, cuando hay que compaginar esa tarea con la integración de modificaciones en la aplicación legada, o resolver incidencias encontradas en las pruebas de usuario.
- **Silk Central**[19]**:** herramienta de la empresa Micro Focus para la gestión y seguimiento de planes de prueba. Puede sincronizarse con la anterior y, al igual que ella, es de especial utilidad en las fases que implican construcción de software y realización de pruebas. Permite conocer el grado de cobertura de requisitos y el porcentaje de avance en la ejecución de casos de pruebas. Esta información puede consultarse online y plasmarse en informes en diferentes formatos. Ofrece también la posibilidad de diseñar informes particulares para cada proyecto.
- **ArchiMate**[20]**:** herramienta open source para el modelado de arquitecturas que admite la descripción, el análisis y la visualización de éstas en distintas áreas de negocio. De gran utilidad en las fases de estudio de aplicaciones para obtener una visión general de su estructura e interacción con el entorno.
- **Microsoft Project**[21]**:** herramienta para la gestión de proyectos. Su utilidad no se circunscribe a determinadas fases de desarrollo del proyecto. Facilita el seguimiento del avance, la detección temprana de posibles desviaciones de tiempo y cargas de trabajo inadecuadas.

[18] https://www.atlassian.com/es
[19] https://www.microfocus.com/es-es/media/data-sheet/silk_central_ds_es.pdf
[20] https://www.archimatetool.com/
[21] https://www.microsoft.com/es-es/microsoft-365/project/project-management-software

4.3. Documentación

En el manifiesto ágil se defiende la prevalencia del software que funciona frente a la documentación y una interpretación sesgada de este principio puede hacer pensar que en las metodologías ágiles se descarta la generación de todo tipo de documentación. Sin embargo, si uno de los problemas que se atribuye al mantenimiento de los sistemas legados es la falta de documentación, no sería apropiado proponer un nuevo método de trabajo que abogara por la total desaparición de esta. El punto intermedio es generar la documentación precisa para facilitar el desarrollo y mantenimiento de la aplicación, así como la formación de los componentes del equipo encargado de las mencionadas tareas. Es decir, se debe contar con una documentación suficiente y actualizada que sea, además, fácil de mantener.

En el método de trabajo propuesto no se contempla una fase específica para generar documentación, sino que se considera una parte más del proceso de desarrollo. Se elabora de manera iterativa al mismo tiempo que se construye la nueva aplicación y se adapta a los cambios que se van produciendo. De esta manera, cuando finaliza el proyecto se cuenta con documentos que reflejan perfectamente el estado de la aplicación en ese momento.

Parte de la documentación que se propone generar coincide con la que se elaboraría en cualquier proyecto de desarrollo de software y otra es específica de un proceso de migración:

- Documentación general:
    - Descripción técnica del sistema (diseño técnico, inventario de módulos, base de datos).
    - Documentación de pruebas.
    - Manual de usuario:
        - Funcional, para el usuario final.
        - Técnico, para otras aplicaciones que necesiten utilizar elementos interfaz.
- Documentación propia de la migración:
    - Documento de correspondencia entre campos de BBDD legada y nueva, y reglas de transformación.

- Documento de correspondencia entre módulos interfaz legados y nuevos utilizados por otras aplicaciones.

4.4. Equipo de trabajo

El modelo de desarrollo en cascada suele asociarse con un equipo de estructura jerárquica poco flexible, liderado por un jefe de proyecto sobre el que recae toda la responsabilidad del devenir del proyecto y que dirige a una serie de analistas y programadores de diferente experiencia y capacidad. El método de trabajo propuesto mantiene el desarrollo en cascada, pero al mismo tiempo sugiere introducir elementos de metodologías ágiles. En el ámbito de la estructura del equipo de trabajo esto va a suponer un nuevo reparto de responsabilidades y la inclusión de actores con los que no se suele contar en los desarrollos tradicionales.

En lo referente al nuevo reparto de responsabilidades, se propone una descomposición de la aplicación en subsistemas que permita establecer distintas líneas de trabajo paralelas y ceder el liderato de cada una de ellas a los analistas del equipo. Cada uno de ellos trabajará con el subconjunto de programadores que le haya sido asignado para el desarrollo de un determinado subsistema y este subequipo, aun manteniendo el modelo de trabajo en cascada, tendrá autonomía en lo referente a quién desarrolla una determinada tarea, o cuándo se aborda la construcción de cada uno de los elementos que componen el subsistema del que son responsables. Es decir, se reproduce una estructura y una dinámica de trabajo similar a la que rige en los equipos Scrum.

Continuando en este acercamiento al modelo Scrum y ya en el plano de los nuevos roles que deben ser tenidos en cuenta, se propone trabajar con una figura similar a la del Product Owner, entendido éste como la persona o conjunto de personas conocedoras de la funcionalidad que se debe implementar y encargadas de dar el visto bueno a las implementaciones que se van realizando. Este papel podría estar desempeñado, por ejemplo, por un usuario final y su rol sería el de Responsable funcional.

Existe otra figura que el modelo Scrum no contempla como rol, pero sí prevé su participación en algunas de las ceremonias propias de esa metodología. Se trata del stakeholder o parte interesada. Alguien que, sin estar totalmente integrado en el desarrollo, se ve afectado por el proyecto o puede aportar al mismo algún

tipo de conocimiento. En el caso de una migración podría quedar representada por alguno de los componentes del equipo responsable del mantenimiento de la aplicación legada y su rol sería el de Colaborador ocasional.

En este nuevo escenario, el jefe de proyecto tiene la responsabilidad de coordinar y supervisar todas las líneas de trabajo, y de liderar la interlocución con el Responsable funcional y los posibles Colaboradores ocasionales, asumiendo así un papel similar al del Scrum Master.

Gráficamente, la estructura del equipo quedaría representada en la Figura 11.

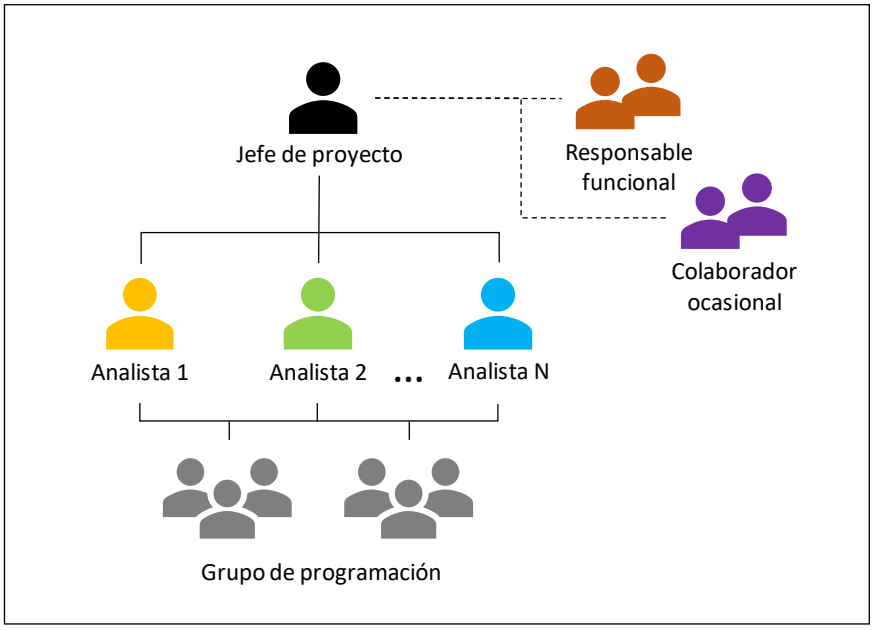


Figura 11 – Representación gráfica de la estructura de equipo propuesta

Fuente: elaboración propia.

La composición del equipo en general, y el número de analistas en particular, está condicionada al número de líneas paralelas de desarrollo que se quieran (o puedan) mantener y al margen de fechas de trabajo que se haya establecido. El dimensionamiento del grupo de programación está afectado por esos mismos factores, al que hay que añadir el hecho de que la asignación de sus miembros a cada una de las líneas de trabajo no es fija a lo largo del proyecto, sino que depende de la carga de trabajo que éstas generen en cada momento del desarrollo.

4.5. Estructura del modelo de trabajo propuesto

Al igual que las metodologías presentados en el capítulo anterior, el modelo de trabajo que aquí se propone se estructura en fases. Con ellas se pretenden cubrir los objetivos de tamaño, metodología, complejidad, evolución y cambios, y documentación que se han expuesto anteriormente y evitar en lo posible los problemas ya conocidos asociados a las migraciones de aplicaciones Mainframe. Algunas de las fases tienen similitudes con las que se definen en metodologías ya conocidas y otras son nuevas, conformando el elemento diferenciador de esta propuesta.

4.5.1. FASE 1 – Estudio del entorno y la tecnología destino

Un punto importante que no suelen tener en cuenta las metodologías estudiadas hasta ahora es la adquisición, por parte del equipo encargado de la migración, del conocimiento técnico del entorno destino antes de comenzar el estudio de la aplicación que se pretende migrar. Este conocimiento, o bien suele darse por supuesto y se vuelca todo el esfuerzo en el estudio de la aplicación a migrar, o bien no se repara en la conveniencia de conocer adecuadamente el terreno donde deberá construirse la nueva aplicación antes de iniciar el estudio de la que se migra. Como excepción puede mencionarse la metodología XIRUP, que en su etapa “Comprensión” incluye la plataforma destino, las transformaciones necesarias y las restricciones que limitan la construcción del nuevo sistema en la información que debe ser recopilada.

Como se verá más adelante, el estudio de la aplicación migrada no consiste únicamente en asimilar la funcionalidad del negocio que implementa, supone también estudiar las soluciones técnicas que se tomaron en su día para cumplir con requisitos funcionales y no funcionales y dar respuesta a las peticiones particulares de los usuarios finales. Conocer si es factible o no construir este tipo de particularidades en el nuevo escenario respetando las normas de convivencia que establece la propia infraestructura ayuda a anticiparse a los posibles problemas y a no pasar por alto ningún detalle. De ahí la importancia de que la etapa dedicada al conocimiento del entorno destino vaya en primer lugar.

Dentro de este estudio previo y teniendo en cuenta que el destino es un entorno Mainframe, los puntos a los que se debe prestar especial atención son:

- Arquitectura.
- Patrones de construcción.
- Base de datos.
- Aplicaciones corporativas o de infraestructura.

4.5.1.1. Arquitectura.

La implantación de una arquitectura Mainframe está detrás de muchas de las migraciones de los sistemas de información de grandes corporaciones, como bancos, compañías de seguros, etc. Estas arquitecturas establecen una serie de reglas y buenas prácticas que condicionan el desarrollo y ejecución de las aplicaciones que se encuentran bajo su dominio. Seguir estas pautas facilita la interacción con las aplicaciones que ya están integradas en la arquitectura y simplifica la construcción de las nuevas, estandarizando el desarrollo y el mantenimiento del software. Conocerlas con antelación da la oportunidad de anticiparse a la estructura y modularización de la nueva aplicación, facilita el estudio del código legado al permitir obviar ciertas partes de él, y da pie a mejorar la funcionalidad de la aplicación integrando funciones o herramientas de la arquitectura destino. Como ejemplos de lo anterior pueden mencionarse los siguientes casos:

- Uno de los problemas que suele presentar el código legado es su carácter monolítico. Las arquitecturas Mainframe solucionan en parte este problema ofreciendo una estructura en capas, lo que permite (por ejemplo) separar las funciones de presentación, negocio y datos. Tener presente esta organización durante el estudio de los elementos de la aplicación legada permite anticipar la estructura y modularidad de la nueva.
- Si la arquitectura en la que debe integrarse la nueva aplicación asume determinadas funciones que están presentes en todos o en la mayoría de los módulos, la cantidad de código legado que debe estudiarse con detalle puede verse reducida considerablemente. Tomando como ejemplo la gestión de errores o el intercambio de información con el usuario a través de una interfaz, podría obviarse todo lo referente a cómo se capturan y controlan los errores o no prestar atención a las sentencias propias del envío y recepción de información hacia/desde el usuario.

4.5.1.2. Patrones de construcción.

Gran parte de las funciones que implementan las aplicaciones Mainframe responden a acciones de tipo CRUD (Crear, Leer, Actualizar y Borrar). Aunque esta funcionalidad deba completarse con las validaciones y reglas de negocio que marque cada aplicación, la arquitectura puede ofrecer patrones de construcción que respondan a los flujos de funcionamiento más habituales y facilitar la homogeneización tanto del código como de la estructura de las aplicaciones. Al conocer de antemano cuáles y cómo son los patrones a los que habrá que ajustarse durante el desarrollo, se puede prever si alguna de las funcionalidades de la aplicación legada no encajará en ellos. De esta manera se puede anticipar la aparición de problemas de diseño y comenzar cuanto antes a trabajar en las posibles soluciones.

4.5.1.3. Base de datos.

No son demasiados los gestores de bases de datos (BBDD) que habitualmente se utilizan en entornos Mainframe. Los cambios más importantes que pueden darse en este aspecto es el paso de un sistema de ficheros al uso de una BBDD (poco frecuente ya en la actualidad), o la evolución de un modelo jerárquico a otro relacional, además del más que posible cambio de formato de datos estándar como fechas, horas o timestamps. El cambio de modelo va a influir tanto en el diseño de la BBDD como en la lógica que se utilice para el acceso a las nuevas tablas. Nuevamente la ventaja que proporciona la capacidad de anticipación ante los problemas que puedan surgir con posterioridad será de gran ayuda.

4.5.1.4. Aplicaciones corporativas y de infraestructura.

Dentro de un entorno Mainframe existen aplicaciones que centralizan la gestión de determinada información general y de uso común, como puede ser la gestión de calendarios y fechas, tipos de monedas, información de clientes, definición de productos comerciales, etc. A este tipo de aplicaciones se las conoce como aplicaciones corporativas. Además de ellas existen las aplicaciones de infraestructura, que son aquellas que conforman el entramado técnico que sustenta a las aplicaciones de gestión. Por ejemplo, pueden implementar

funcionalidades para la gestión de listados, formateo y envío de correos, gestión de mensajería estándar, etc.

Unas y otras liberan a las aplicaciones de negocio de ciertas funcionalidades simplificándolas en gran medida. A la hora de comenzar con una migración es importante saber si habrá aplicaciones corporativas y de infraestructura al servicio del desarrollo y, en caso de ser así, conocer cuáles son. Sobre todo, es importante saber si el escenario varía con respecto al que existía en el sistema origen, es decir, si aparecen o desaparecen aplicaciones corporativas y de infraestructura con respecto a lo anterior, o si éstas cambian su funcionalidad de alguna manera. El hecho de que existan nuevas aplicaciones corporativas o de infraestructura puede evitar la implementación de algunas de las funcionalidades codificadas dentro de la aplicación migrada. Por otro lado, la desaparición de este tipo de aplicaciones puede significar que la funcionalidad que ofrecían ha pasado a ser obsoleta o que alguna otra de nueva aparición ha asumido su papel. En ambos casos habrá que confirmarlo para tener una idea clara del escenario en el que se va a trabajar.

4.5.2. FASE 2 – Determinación del alcance funcional de la migración

A priori, el alcance del proyecto sería poco discutible, ya que por definición todo lo que esté contemplado en la aplicación origen debería quedar implementado en la nueva que se construye. Sin embargo, no es inusual aprovechar los proyectos de migración para eliminar funcionalidades obsoletas y mejorar aquellas otras que sí permanecerán. El objetivo de ello no es otro que aumentar la satisfacción del usuario.

En la migración de aplicaciones con largo recorrido en producción (como lo son la gran mayoría de las que se migran en entorno Mainframe) definir el alcance basándose únicamente en la funcionalidad que implementan sin hacer ningún tipo de revisión no es del todo recomendable. De ser así, probablemente se arrastren a la nueva aplicación funcionalidad y código innecesarios, haciendo que nazca ya con cierto grado de obsolescencia. La finalidad de esta segunda fase es detectar (si las hay) qué funcionalidades son obsoletas, cuáles deben permanecer tal y como están funcionando en ese momento, y qué otras podrían mejorarse. Como ejemplo de estas últimas puede mencionarse, por ejemplo, una excesiva navegación entre menús en alguna de las opciones, la obligación de

teclear información fácilmente recuperable desde alguna BBDD, o la presentación de informes con un formato que resulte poco práctico para el usuario.

La importancia de la segunda fase reside en establecer el alcance real del proyecto de la manera más ajustada posible para no presupuestar tareas que realmente no sean necesarias ni dejar atrás otras que sí deban ser tenidas en cuenta. Lo primero daría lugar a una percepción incorrecta del coste del proyecto y lo segundo llevaría (muy probablemente) al incumplimiento del presupuesto inicialmente calculado.

4.5.3. FASE 3 – Definición de subsistemas dentro de la aplicación

Como ya se ha comentado anteriormente, cuanto más grande sea un proyecto, más complicado resulta finalizarlo en la fecha acordada, dentro del presupuesto y con satisfacción del cliente. Según esa premisa, resulta más factible finalizar exitosamente varios proyectos pequeños que obtener ese mismo resultado con un proyecto que aglutine la funcionalidad conjunta de todos ellos. Extrapolando este principio al ámbito del modelo de trabajo que aquí se propone, la manera de conseguir un modelo de pequeños proyectos que abarquen toda la funcionalidad de una aplicación Mainframe es su descomposición en subsistemas e independizar después la gestión de cada uno de ellos. Es decir, la aplicación pasa a ser considerada un sistema de software en lugar de un todo. Las fechas de inicio y fin de cada unidad de gestión o subsistema son independientes entre sí, aunque todas quedan supeditadas a los límites globales del proyecto en su conjunto.

El número y el tamaño de estos subsistemas va a depender de las características propias de cada aplicación y de las funcionalidades que implemente, aunque existen algunos con los que siempre habrá que trabajar y que se recomienda, además, que sean los primeros en abordarse por las dependencias que los demás presentan con respecto a ellos. Estos son:

- Subsistema Base de datos.
- Subsistema Interacción con otras aplicaciones.
- Subsistema Elementos de uso común.

4.5.3.1. Subsistema Base de datos

Comprende el diseño y construcción de la nueva BBDD, así como la migración de los datos legados.

- Diseño y construcción de la nueva BBDD

  A partir del estudio de la BBDD legada (tablas, índices, formato de los datos, etc.) y ateniéndose a los condicionantes técnicos del gestor de BBDD destino se diseñan y construyen las nuevas tablas, se asignan índices y se define cualquier otro tipo de entidad relacionada con la BBDD. Este paso sirve también para localizar (si los hay) aquellos datos que por algún motivo no serán de utilidad en la nueva a aplicación (obsoletos), incluir la definición de alguno nuevo y detectar posibles cambios de formato.

- Migración de datos

  Una vez que se cuenta con la BBDD física, debe poblarse con los datos provenientes del sistema legado. Para ello, el primer paso es establecer la correspondencia de campos y determinar con exactitud el origen de todos los datos que llegan a la BBDD de la nueva aplicación, y el destino de los que salen de la BBDD legada. Para los casos en los que el formato del dato origen no coincida con el asignado en destino se deben elaborar reglas de transformación que conviertan de uno a otro de manera unívoca. Salvo excepciones propias de la aplicación, estas reglas deben ser de aplicación universal, es decir, todos los datos de un mismo tipo deben sufrir el mismo tipo de transformación antes de ser añadidos al nuevo sistema.

Es importante contar cuanto antes con la BBDD para que cuando se comience con el desarrollo de los nuevos módulos se pueda acceder a ella. Además, tener documentada la correspondencia entre campos y las reglas de transformación facilita la tarea de codificación.

4.5.3.2. Subsistema Interacción con otras aplicaciones

Este subsistema recoge los elementos que forman parte de la interfaz con otras aplicaciones. Por lo general suelen ser módulos que actúan a modo de servicios a través de los cuales la aplicación legada comparte información con otras del

entorno. Se deben estudiar los datos que se intercambian y, en su caso, adaptar las interfaces con las aplicaciones con las que se comunica.

La importancia de abordar este subsistema lo antes posible reside en que cuanto antes conozcan las aplicaciones los ajustes que deben realizar para adaptarse a la aplicación migrada, más tiempo tendrán para planificarlos e implementarlos, disminuyendo de esta manera las probabilidades de retraso en la implantación.

Como acción intermedia para adelantar en lo posible la disponibilidad de las nuevas interfaces a las otras aplicaciones, se pueden encapsular los módulos legados y posteriormente realizar las transformaciones internas que sean necesarias.

4.5.3.3. Subsistema Elementos comunes

Este subsistema lo conforman los elementos que son de uso común en toda la aplicación. Por lo general suelen ser módulos encargados de validaciones frecuentes, cálculo de dígitos de control, formateo de campos, etc. El desarrollo de la nueva aplicación se agiliza si se cuenta con ellos en el momento en el que comienza la fase de codificación.

4.5.4. FASE 4 – Estudio de la aplicación legada

El estudio se centra en los subsistemas y funcionalidades recogidos en el alcance del proyecto, y en las particularidades técnicas que pueda presentar la aplicación. Las metodologías descritas en el capítulo anterior incluyen una etapa de adquisición de conocimientos funcionales relativos a la aplicación origen, pero no reparan (o al menos no lo hacen de manera explícita) en la doble vertiente funcional-técnica que aquí se propone.

El **conocimiento funcional** supone la asimilación de las reglas de negocio que conforman el funcionamiento de la aplicación y que habrá que replicar en la nueva que se construya. No se pueden variar porque supondría un cambio significativo, pudiendo llegar incluso a contradecir alguna normativa en la que estuviera basada la aplicación.

Las **particularidades técnicas** son aquellas soluciones concretas que se han implementado para dar una respuesta técnica a determinados requisitos funcionales específicos de la aplicación. Son puntos singulares de la codificación

que ponen en alerta sobre las dificultades que podrían aparecer durante la programación de la nueva aplicación. Dicho de otra manera, evidencian la falta de solución estándar a un determinado problema que habrá que resolver también en el sistema destino, donde probablemente tampoco se cuente con un procedimiento estándar para ello. Como muestra de este tipo de situaciones se podría mencionar (por ejemplo) una gestión de confirmación de operaciones. Lo habitual es que la confirmación se pida una única vez al usuario: se genera una operación y, previamente a su envío y/o consolidación en el sistema, se le pregunta al usuario si está de acuerdo con ese determinado movimiento. Se le da la opción de pulsar “Sí” para continuar adelante con él, o “No” si lo que se desea es abortar la operación. Al ser este el funcionamiento habitual, el sistema destino está preparado para dar una solución estándar a este problema. Sin embargo, es posible que por el carácter especial de una determinada operación se quiera solicitar al usuario dos confirmaciones: una particular sobre un dato concreto de la operación, y otra general sobre toda ella. En ese caso será necesario buscar una solución alternativa que permita dar al usuario lo que necesita sin salirse de las reglas y buenas prácticas que rigen en el sistema destino. Aunque la solución concreta se diseñe e implemente en una fase posterior, es importante ser consciente cuanto antes de la dificultad que esto vaya a suponer.

Para este estudio se cuenta principalmente con tres fuentes de información:

- Equipo de mantenimiento de la aplicación legada.
- Documentación de la aplicación legada.
- Código fuente de la aplicación legada.

4.5.4.1. Equipo de mantenimiento de la ampliación legada

Es el primer eslabón en la cadena de toma de contacto con la aplicación que se quiere migrar. Son los encargados de mantenerla y quienes mejor conocen su situación actual. Dentro de su participación en el proyecto, su colaboración en esta etapa de adquisición de conocimiento sería la de impartir sesiones formativas a los componentes del equipo de migración, actualizar y poner a su disposición la documentación que exista sobre la aplicación y señalar los elementos singulares que deberían ser estudiados en detalle en esta etapa para

anticiparse a los problemas de implementación que pudieran aparecer más adelante.

A la hora de solicitar la colaboración del equipo de mantenimiento de la aplicación legada puede encontrarse cierta resistencia al cambio, fundamentado principalmente por la preocupación de no ser necesarios una vez que se haya convertido la aplicación al nuevo escenario tecnológico. Una manera de evitar este inconveniente podría ser integrarlos en el equipo de desarrollo de la nueva aplicación (o al menos a alguno de ellos). La colaboración del equipo de mantenimiento es una condición clave para conseguir el éxito del proyecto.

4.5.4.2. Documentación de la aplicación legada

Uno de los principales problemas que se han señalado relativos a las aplicaciones legadas es su carencia de documentación, pero al mismo tiempo se argumenta como gran inconveniente que en los sistemas tradicionales todo debe quedar documentado con el mayor detalle posible. De aquí se deduce que el verdadero problema no es que no exista documentación, sino que la que hay posiblemente no esté actualizada.

El primer paso es conocer de qué documentación se dispone y cuál es el grado de desactualización que presenta. A partir de ahí se determina qué manuales o qué parte de ellos son aprovechables y se pueden utilizar para la adquisición de conocimiento.

4.5.4.3. Código fuente de la ampliación legada

El último recurso para el estudio inicial de la aplicación legada es el análisis de su código fuente. Si bien en esta fase se busca más un conocimiento general de la funcionalidad evitando entrar en detalles demasiado concretos de la misma (que serán estudiados más adelante), sí es el momento de consultar los elementos singulares que hayan sido señalados por el equipo de mantenimiento de la aplicación y estudiar las particularidades que podrían ser fuente de problemas en alguna fase posterior.

4.5.5. FASE 5 – Estimación del esfuerzo, planificación del desarrollo completo y dimensionamiento del equipo.

Una vez determinado el alcance del proyecto y habiendo adquirido un conocimiento suficiente tanto de la funcionalidad como de las peculiaridades técnicas de la aplicación legada, se realiza una estimación del esfuerzo en horas que supondrá la construcción de la nueva aplicación. Al tratarse de un proyecto de migración, la aplicación legada seguirá evolucionando mientras se lleva a cabo la construcción de la nueva, factor que también habrá de tenerse en cuenta a la hora de determinar el esfuerzo total. Para estimar el tiempo extra que supondrá la integración de esas modificaciones a lo largo del desarrollo se puede recurrir a la media de horas anuales dedicadas a las modificaciones funcionales de la aplicación legada en un determinado período de tiempo.

Otro aspecto que debe quedar fijado en esta fase es la duración máxima del proyecto y el emplazamiento temporal dentro del calendario de trabajo. Es decir, se trata de determinar el margen de fechas de inicio y fin en las que se tendrá que desenvolver el proyecto.

Conocidos los márgenes temporales y el esfuerzo necesario para la construcción de la nueva aplicación, llega el momento de dimensionar el equipo. Si bien el proyecto de migración engloba la transformación de la aplicación al completo, el desarrollo se plantea abordando la construcción de subsistemas uno por uno. Esto da la posibilidad de trabajar en paralelo en varios de ellos y acortar así la duración total del proyecto. Cada una de estas líneas de trabajo tendrá asignada una parte del equipo, estará liderada por un analista y supervisada por el jefe de proyecto.

La última tarea que se afronta dentro de esta fase es la planificación completa del proyecto. Conocidos los subsistemas que habrá que construir, el equipo con el que se cuenta, las líneas de trabajo que se pueden abrir y las fechas de inicio y fin de proyecto, se debe establecer un plan de trabajo que permita cumplir con los objetivos marcados. Al final de esta fase se conocen las fechas de inicio y fin del desarrollo de cada uno de los subsistemas, su volumen de horas y los componentes del equipo que participarán en su construcción.

4.5.6. FASE 6 – Proceso iterativo de construcción de subsistemas

Esta fase engloba las tareas que se deben desempeñar para lograr la construcción de un subsistema y se repetirá para cada uno de aquellos de los que conste la aplicación. Al contemplarse el desarrollo en paralelo de varios subsistemas, es factible tener en vuelo varios procesos de construcción de manera simultánea.

Las etapas de las que consta esta fase responden a un modelo en cascada (diseño, programación, pruebas…), pero la dinámica de trabajo asimila técnicas de metodología ágil, como ya se vio en el capítulo dedicado a la composición y estructura del equipo de trabajo:

- El conjunto de personas responsable de la construcción de un subsistema se autogestiona, determinando quién hace qué, y cómo y cuándo debe llevarse a cabo.
- El jefe de proyecto actúa como coordinador o supervisor de las diferentes líneas de trabajo y ejerce de interlocutor con el responsable funcional y los posibles colaboradores ocasionales, quienes también están involucrados en esta fase del desarrollo.

Volviendo a las etapas que deben cubrirse en esta fase, son las que se indican a continuación y quedan representadas en la Figura 12:

1. Elaboración del diseño técnico del subsistema
2. Construcción y entrega del subsistema
3. Aceptación del subsistema

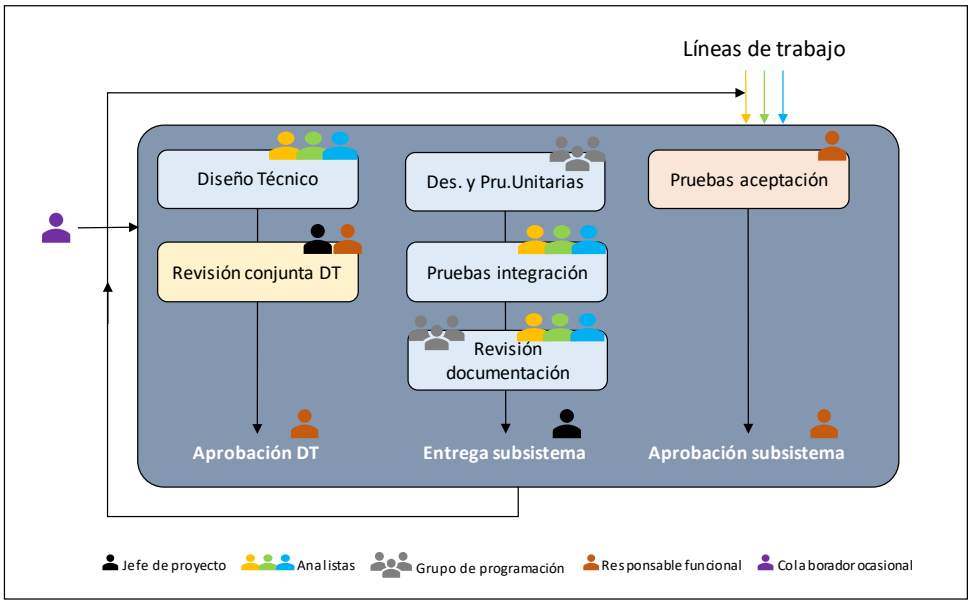


Figura 12 – Representación gráfica de la FASE-6

Fuente: elaboración propia.

4.5.6.1. Elaboración del diseño técnico del subsistema

Es en este punto cuando se realiza un estudio detallado de la funcionalidad que debe implementar el subsistema que se pretende construir. El motivo para ello es que desde que se hizo el estudio preliminar hasta el momento en que se comienza a trabajar en un determinado subsistema, la aplicación legada ha podido sufrir algún cambio, ya sea por ampliación, reducción o modificación de la funcionalidad que presentaba al inicio del proyecto y no tendría mucho sentido desarrollar un subsistema obsoleto desde el inicio. El tiempo utilizado para su construcción más el que habría que dedicar posteriormente para adaptarlo al estado real de la aplicación en producción sería mayor que el que puede suponer el estudio detallado de la situación actual más un desarrollo acorde con el resultado de dicho estudio.

El equipo de desarrollo es el encargado de elaborar el diseño técnico del subsistema. Una vez terminado se entrega al Responsable funcional para su estudio y aprobación.

Posteriormente, a petición del responsable funcional, se convoca una reunión de revisión de diseño técnico donde se consensua el contenido por ambas partes. Una vez aprobado, quedan cerradas las especificaciones. Cualquier modificación posterior se gestionará por el procedimiento de control de cambios (siguiente fase del método de trabajo que se propone).

4.5.6.2. Construcción y entrega del subsistema

Ya con el diseño técnico aprobado se aborda el desarrollo del subsistema y se completan sus pruebas unitarias y de integración. También se generan los documentos de pruebas que permitan reproducir los casos ejecutados y verificar la correcta finalización de todos ellos (documentación interna). Una vez hecho esto se prepara la documentación que acompañará a la entrega del subsistema que se ha construido (documentación externa): se revisan los inventarios, se actualiza el diseño técnico en caso de ser necesario y se elabora la parte correspondiente del manual de usuario.

El hito final de la etapa es poner el subsistema que se ha construido a disposición del responsable funcional. Esto incluye tanto el despliegue del software en el entorno de pruebas de aceptación, como la entrega de la documentación externa que se ha generado.

4.5.6.3. Aceptación del subsistema

Esta etapa es competencia del responsable funcional, quien realiza las pruebas oportunas para dar su aprobación al subsistema recibido. En caso de detectar algún funcionamiento incorrecto, es decir, algo que no se ajuste a lo acordado en el diseño técnico, debe solicitar una corrección.

El desarrollo en paralelo de varios subsistemas puede provocar que todo o parte del equipo que participó en el desarrollo de aquel que se está probando se encuentre trabajando en la construcción de otro. Por ese motivo, las incidencias o modificaciones propuestas por el responsable funcional entran en el proceso de gestión de cambios y se irán abordando de acuerdo con las prioridades y dedicaciones existentes en cada momento. De esta manera se controla la interferencia que la adaptación de subsistemas ya entregados pueda tener en el desarrollo de los nuevos y se evitan posibles retrasos en la entrega de estos últimos. La etapa finaliza con la aceptación del subsistema por parte del responsable funcional una vez acabadas sus pruebas y corregidos todos los problemas reportados.

4.5.7. FASE 7 – Gestión de cambios

Esta etapa comienza en el momento en que se entrega al responsable funcional el primero de los subsistemas construidos y finaliza cuando se da por finalizado el desarrollo de la nueva aplicación al completo. Es decir, no se cierra hasta que todos los subsistemas están construidos y no quedan modificaciones que integrar en la aplicación que se construye.

En esta gestión de cambios se tratarán tanto las incidencias que detecte el responsable funcional en la etapa de pruebas de aceptación de la FASE 6, como las adaptaciones que sea necesario aplicar a los subsistemas ya entregados debidas a la evolución de la aplicación legada que todavía permanece en producción.

La implementación de un cambio incluye la modificación y pruebas del código afectado, así como la actualización de la documentación relacionada, ya sea el inventario, el diseño técnico o el manual de usuario.

Para diseñar el proceso de gestión de cambios se propone tomar como base el modelo ITIL sin seguirlo de manera estricta[22]. Por ejemplo, el Comité de Cambios puede reducirse a una reunión entre el jefe de proyecto y el responsable funcional y no es necesaria la existencia del Consejo asesor del cambio. Como alternativa a este último queda abierta la posibilidad de solicitar la colaboración de expertos en algún tema concreto en el caso de que se considerase conveniente.

Volviendo al modelo ITIL, los pasos que conforman su proceso de gestión de cambios son:

- RFC o solicitud del cambio
- Registro
- Aceptación
- Filtrado y clasificación
- Planificación
- Construcción
- Prueba
- Implementación
- Evaluación

[22] El proceso de “Gestión de Cambios de ITIL v3 tiene su equivalencia en la práctica “Change enablement” de ITIL v4 (https://yasm.com/wiki/en/index.php/ITIL_4_vs_ITIL_V3).

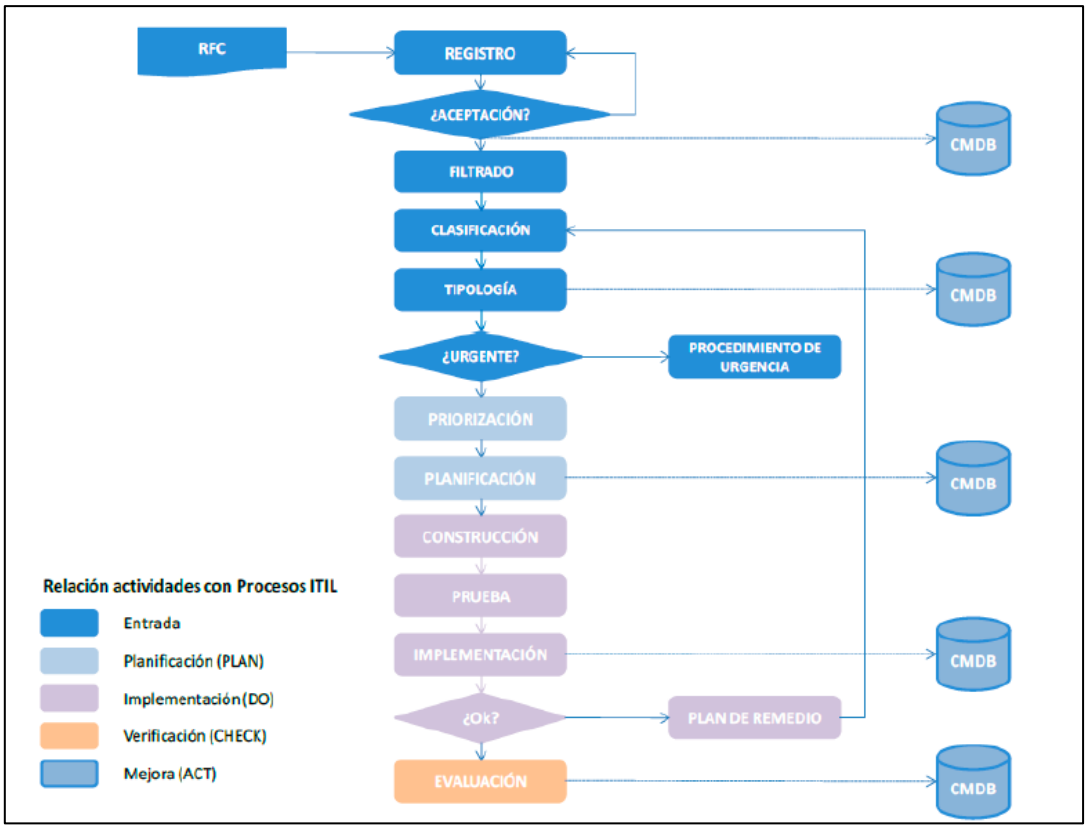


Figura 13 – Actividades del proceso de Gestión de Cambios.

Fuente: Manual de ITILv3 (Ríos Huércano)

En cuanto al proceso a seguir, el responsable funcional será quien dé de alta la solicitud de cambio con la naturaleza que corresponda (incidencia, nueva funcionalidad, etc.) y quien evalúe una vez implementado si cumple las especificaciones o no. El jefe de proyecto se encargará de la gestión del resto de tareas. En terminología ITIL:

- **Propietario del cambio** = Responsable funcional.

  Persona que propone un cambio y, en la mayoría de las ocasiones, se responsabiliza de iniciar la RFC.
- **Gestor de cambios** = Jefe de proyecto

  Es el propietario del proceso y se responsabiliza de la Gestión de Cambios. Asesora al propietario del cambio, acepta y clasifica las peticiones de cambio (RFC), revisa los cambios implementados y convoca las reuniones para la toma de decisiones sobre los cambios a abordar.
- **CMDB** = Sistema de gestión de incidencias y mejoras que se haya decidido utilizar en el desarrollo del proyecto.

4.5.8. FASE 8 – Elaboración del plan de implantación y puesta en producción

Si bien la propuesta para el desarrollo es plantear un sistema iterativo con el que ir construyendo la aplicación por partes, se considera más conveniente que la entrada en producción se haga de toda la aplicación al completo. El motivo es que el desarrollo de interfaces y la continua adaptación tanto del código legado como del nuevo para la integración de ambos sistemas aumentaría el riesgo y alargaría demasiado la duración del proyecto. Lo más conveniente es disminuir la duración de los proyectos el máximo posible para reducir así el número de adaptaciones que haya que integrar en la nueva aplicación por modificaciones hechas en el código legado durante el período de desarrollo. De la misma manera, la transición de una a otra debe llevar el menor tiempo posible y hacerse dentro de una ventana temporal en la que no haya interferencias con modificaciones especialmente importantes de la aplicación que se migra, ni con la entrada en producción de otras con las que ésta se relacione. No es extraño que estas ventanas temporales sean de corta duración, por lo que es recomendable preparar todos los pasos necesarios para la implantación con la mayor antelación posible. Este conjunto de tareas necesarias para preparar la entrada en producción sin incidentes es el plan de implantación. Es de suma importancia no olvidar ninguno de los pasos que deben darse en ese momento. Para ello se propone la creación de un documento que los recoja, que sea sencillo de seguir y que esté consensuado con el responsable funcional de la aplicación. También dentro del plan de implantación se puede contemplar el programa de formación de los componentes del equipo de mantenimiento que asumirá esta tarea una vez que la aplicación comience a ejecutarse en entornos productivos. Con la implantación de la nueva aplicación se pone punto final al proyecto de migración.

5. CASO DE ESTUDIO

5. Caso de estudio: trabajar con y sin el modelo de trabajo propuesto

Una vez expuesto el nuevo método de trabajo cabe preguntarse qué diferencias pueden presentarse en el desarrollo y resultado de un proyecto en el que se aplique y otro en el que no.

Cada una de las fases da solución a inconvenientes o situaciones que se han ido detectando en los distintos proyectos de migración en los que se ha tenido ocasión de participar. En el espacio dedicado a cada una de ellas se comienza exponiendo el problema que había que afrontar y a continuación cómo la propuesta metodológica de este TFM le da solución.

5.1. FASE 1 – Estudio del entorno y la tecnología destino

En los proyectos de migración se había observado que un factor de riesgo que contribuía de manera significativa en la desviación de las fechas de entrega era no conocer de antemano las oportunidades, restricciones y características técnicas que presentaba el entorno destino. Este desconocimiento provocaba soluciones de diseño poco optimizadas y retrasos en el desarrollo del software por tener que rehacer código o por tener que parase a estudiar la manera de resolver un determinado problema técnico cuando ya se había comenzado la codificación. Incluso habiendo dedicado un tiempo inicial a la formación seguían apareciendo estos problemas, bien porque no se contaba con un documento de referencia rápida donde poder consultar las características principales del entorno, bien por no haber cubierto en el estudio algún aspecto importante.

Una alternativa para paliar en lo posible este tipo de inconvenientes es trabajar con un documento tipo check list en el que se recoja el conjunto de puntos mínimos que deben quedar resueltos tras el período de formación inicial, tales como si la arquitectura toma el control del commit y del rollback, cómo es el control de errores, o si existen patrones de construcción y cuál es su obligatoriedad de uso. Como por lo general se busca la estandarización de formato en los documentos de trabajo, así como facilitar su cumplimentación, la utilización de un formulario puede ayudar a cumplir con estos objetivos. En la Figura 14 puede verse un ejemplo de formulario a utilizar en esta primera etapa de la migración.

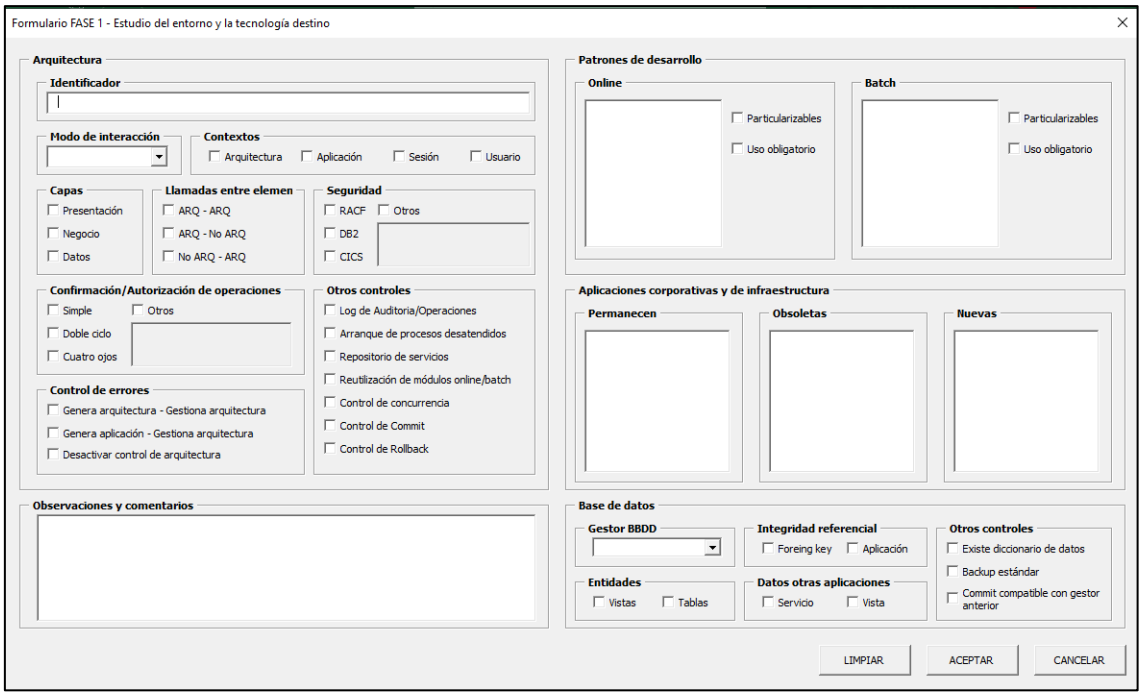


Figura 14 – Formulario propuesto para la FASE-1.

Fuente: elaboración propia

Una vez marcadas las checks correspondientes y rellenados el resto de los campos, al pulsar el botón ACEPTAR se traslada esa información a una hoja Excel preparada para imprimir el informe en formato .pdf (Figura 15). El resultado de este informe cumplimentado para los apartados de BBDD y arquitectura puede verse en las Figuras 16 y 17.

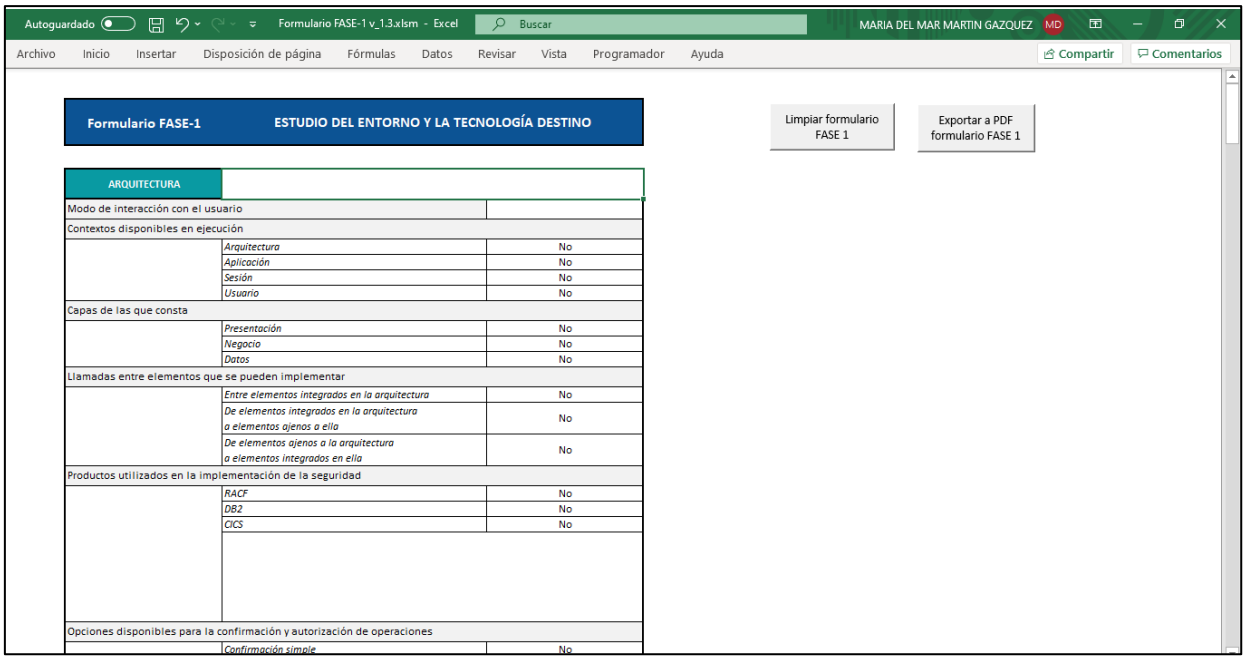


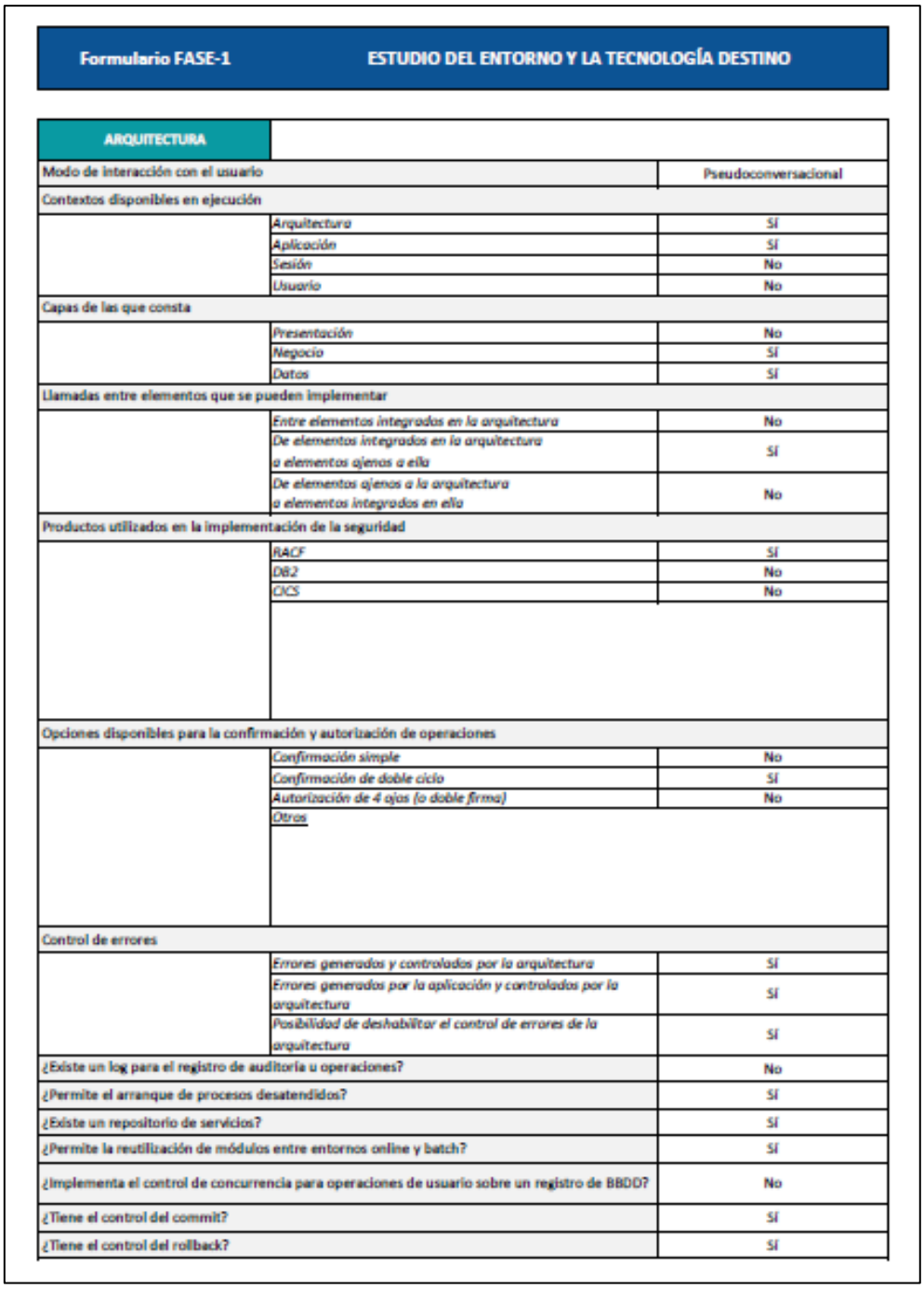
Figura 15 – Información recogida en el formulario de FASE-1 y trasladada a una hoja Excel.

Fuente: elaboración propia

BASE DE DATOS		DDB2
Entidades con las que se trabaja		
	Tablas	Sí
	Vistas	No
¿Cómo implementa la integridad referencial?		
	Mediante Foreing Keys	No
	Mediante programación en las aplicaciones	Sí
¿Cómo se accede a los datos de otras aplicaciones?		
	Mediante llamada a servicios	Sí
	Mediante acceso a vistas	No
¿Existe un diccionario de datos?		No
¿Existe en la instalación un backup estándar de la BBDD?		Sí
El commit de este gestor ¿es compatible con el de la anterior BBDD?		No

Figura 16 – Informe de BBDD obtenido a partir del formulario de FASE-1.

Fuente: elaboración propia



Formulario FASE-1	ESTUDIO DEL ENTORNO Y LA TECNOLOGÍA DESTINO	
ARQUITECTURA		
Modo de interacción con el usuario		Pseudoconversacional
Contextos disponibles en ejecución		
	Arquitectura	Sí
	Aplicación	Sí
	Sesión	No
	Usuario	No
Capas de las que consta		
	Presentación	No
	Negocio	Sí
	Datos	Sí
Llamadas entre elementos que se pueden implementar		
	Entre elementos integrados en la arquitectura	No
	De elementos integrados en la arquitectura a elementos ajenos a ella	Sí
	De elementos ajenos a la arquitectura a elementos integrados en ella	No
Productos utilizados en la implementación de la seguridad		
	RACF	Sí
	DB2	No
	CICS	No
Opciones disponibles para la confirmación y autorización de operaciones		
	Confirmación simple	No
	Confirmación de doble ciclo	Sí
	Autorización de 4 ojos (o doble firma)	No
	Otros	
Control de errores		
	Errores generados y controlados por la arquitectura	Sí
	Errores generados por la aplicación y controlados por la arquitectura	Sí
	Posibilidad de deshabilitar el control de errores de la arquitectura	Sí
¿Existe un log para el registro de auditoría u operaciones?		No
¿Permite el arranque de procesos desatendidos?		Sí
¿Existe un repositorio de servicios?		Sí
¿Permite la reutilización de módulos entre entornos online y batch?		Sí
¿Implementa el control de concurrencia para operaciones de usuario sobre un registro de BBDD?		No
¿Tiene el control del commit?		Sí
¿Tiene el control del rollback?		Sí

Figura 17 – Informe de arquitectura obtenido a partir del formulario de FASE-1.

Fuente: elaboración propia

5.2. FASE 2 – Determinación del alcance funcional de la migración

Otro problema observado en los proyectos de migración era que el usuario no quedaba del todo satisfecho con la nueva aplicación que recibía porque la funcionalidad que le ofrecía no se ajustaba a lo que él realmente necesitaba para su trabajo diario. Es decir, tenía delante una aplicación construida con tecnología

más moderna y que probablemente presentara unas prestaciones técnicas superiores a la anterior, pero en lo relativo a la mejora para el desempeño de su trabajo poco tenía que aportar. Analizando esta situación se llegó a la conclusión de que el origen del problema estaba en una definición inadecuada del alcance del proyecto. Al tener que construir una aplicación equivalente a otra, se solía dar por sentado que del alcance poco se podía discutir. El único requisito era que la nueva aplicación debía replicar el funcionamiento de la original, sin dar importancia al hecho de que el paso del tiempo podría haber dejado obsoleto algún aspecto de la aplicación que se migraba, o que desde su construcción hasta el momento actual hubieran surgido nuevas necesidades a las que por algún tipo de condicionante técnico no se hubiera podido dar respuesta.

Para la definición del alcance se parte habitualmente de la documentación que haya disponible, que suele consistir en documentos de texto voluminosos y no del todo actualizados que por sí mismos no dan una idea general de las funcionalidades que cubre la aplicación legada, la interacción entre ellas, etc. Ante esta situación resulta complicado tener una visión de qué es lo que realmente se está utilizando, qué parte ha caído en desuso o dónde se puede actuar aprovechando la migración para dar valor añadido a la aplicación resultante. Pensando también en las ventajas que ofrece en etapas posteriores, una buena alternativa es utilizar la herramienta ArchiMate para construir el modelo de arquitectura de la aplicación. La vista global (Figura 18) sirve como documento de trabajo para que, tras sucesivas reuniones con usuarios y responsables de la aplicación, se fije un alcance más real del proyecto de migración. Gracias a él se identifican las funcionalidades que han caído en desuso y pueden proponerse mejoras que agilicen y aporten calidad al trabajo realizado por el usuario. Como ejemplo de esto último se podría mencionar la sustitución de listados estáticos que el usuario solicite periódicamente por un sistema de consultas online en tiempo real.

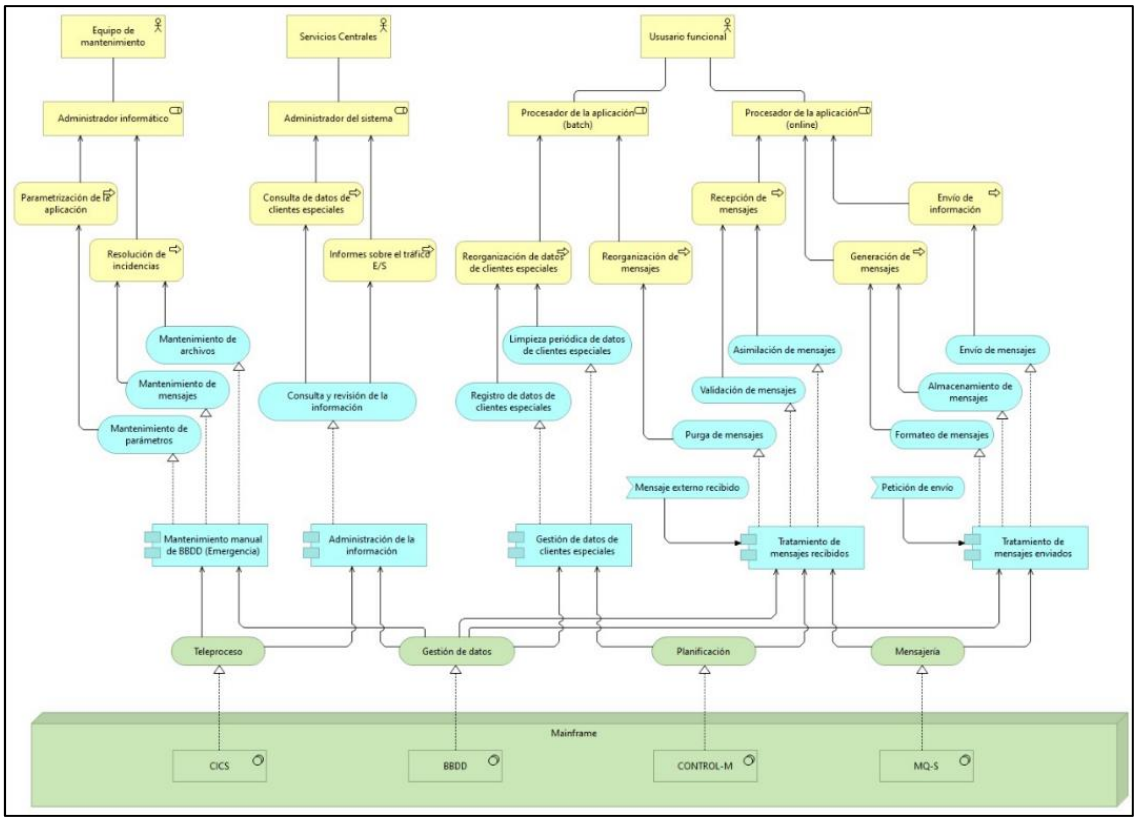


Figura 18 – Ejemplo de diagrama ArchiMate de aplicación legada.

Fuente: elaboración propia

5.3. FASE 3 – Definición de subsistemas dentro de la aplicación

Una vez definido el alcance, queda determinado también el volumen del software a migrar. El reto en esta etapa es encontrar una solución al problema inherente a los proyectos de migración de aplicaciones legadas Mainframe: son de gran tamaño y eso es un importante factor de riesgo para el cumplimiento de fechas y de presupuesto económico.

Teniendo en mente los nuevos modelos de gestión y organización de proyectos como puede ser la metodología Scrum, el planteamiento para evitar este escollo a priori infranqueable es ir dando al usuario acceso a la aplicación por etapas en vez de hacerlo al completo en una única entrega. La manera de poder cumplir con esa propuesta y permitir al mismo tiempo que el usuario realice sus pruebas a medida que va recibiendo funcionalidades es la definición de subsistemas. Es decir, contar unidades de negocio que formando parte de un todo (el proyecto completo) tengan la suficiente autonomía como para poder ser manejadas de manera independiente desde la etapa de diseño técnico hasta la de pruebas de usuario.

El diagrama ArchiMate obtenido en la FASE-2 facilita enormemente esta tarea. El modelo de la arquitectura de la aplicación permite hacer el trabajo de división en subsistemas de manera gráfica (Figura 19).

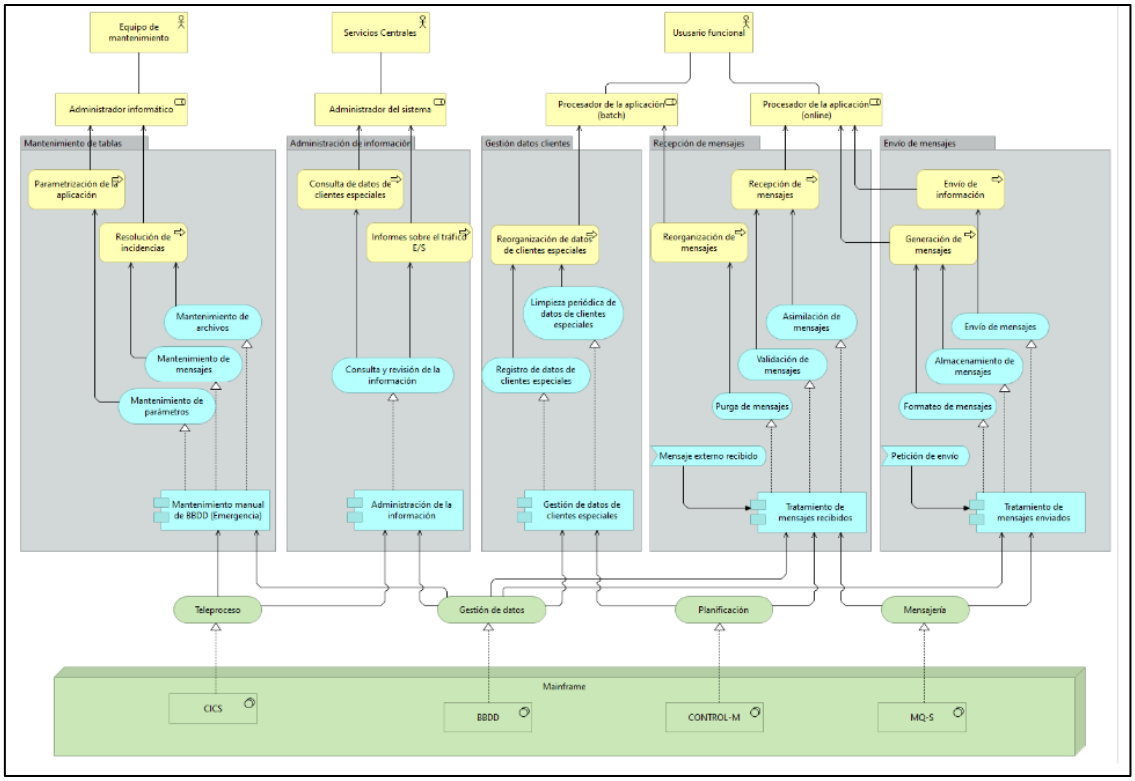


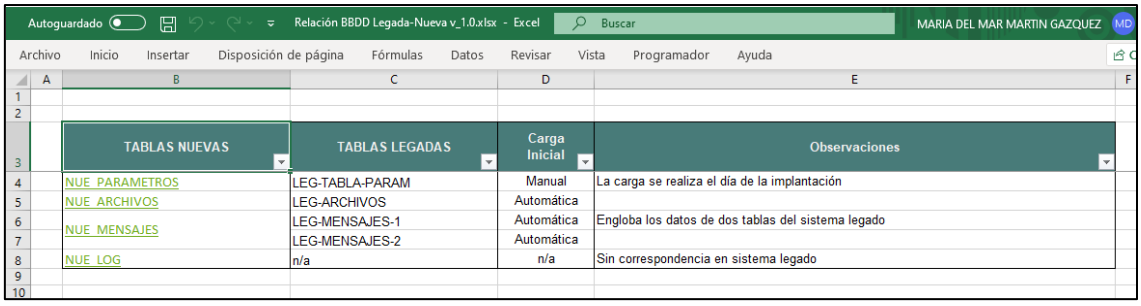
Figura 19 – Ejemplo de diagrama ArchiMate de aplicación legada con subsistemas.
Fuente: elaboración propia

En lo que respecta a los tres subsistemas “estándar” de aplicaciones mainframe (base de datos, interfaces con otras aplicaciones y elementos comunes), al no tratarse de subsistemas funcionales propiamente dichos, no entran dentro de los planes de prueba de los usuarios. Sin embargo, es importante tenerlos en cuenta como entidades independientes y abordar su migración en el momento oportuno. De no hacerse así, es muy probable que se acaben generando retrasos, ya sea por tener que esperar a que se construyan o por hacer adaptaciones posteriores en la aplicación que suponen rehacer código ya probado.

Tradicionalmente, al diseño, construcción y población de la nueva BBDD no se le ha dotado de entidad propia, sino que se ha asumido como parte de las tareas que se llevan a cabo durante el tiempo de codificación y pruebas unitarias de los programas. Es decir, se ha considerado una tarea transversal y no se ha dado importancia a contar con una BBDD consolidada y con datos reales (entendidos

éstos como los heredados de la aplicación legada) hasta el momento de las pruebas de integración. Por ese motivo, frecuentemente los programadores deben crear y probar sus programas sin tener una referencia clara de cuál debe ser el contenido de las tablas o incluso del formato de algunas de sus columnas. Esta circunstancia y la corrección y prueba de los programas que esto provoca durante el período de pruebas de integración, resultan ser ladrones de tiempo y la consecuencia es que las estimaciones de horas para desarrollo y pruebas de integración se quedan cortas y se producen retrasos en la entrega a los usuarios. Para evitar esta situación no deseada se anticipa la migración del subsistema de BBDD. Esta anticipación permite definir la nueva BBDD en todos los entornos (incluido Producción) casi al inicio del proyecto e ir cargándola y depurando datos de manera muy temprana, lo que redunda en una reducción del riesgo de pérdida de datos o de errores de conversión una vez que la nueva aplicación entre en funcionamiento.

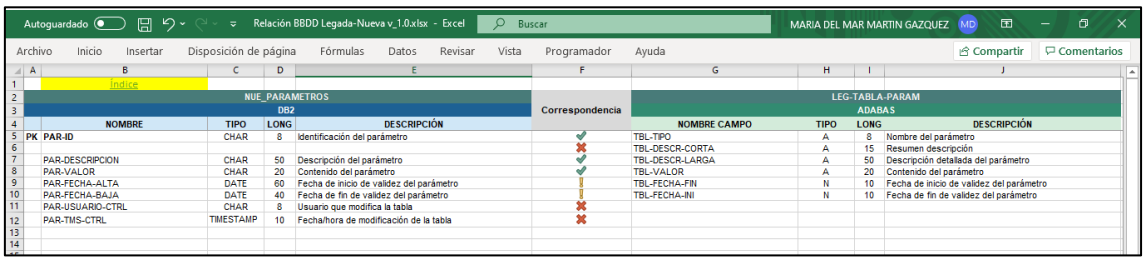
Para facilitar la identificación correcta de las equivalencias entre tablas y columnas de las BBDD legada y nueva se crea, como parte de la documentación de la aplicación, un documento que recoge esta equivalencia (Figuras 20 y 21).



TABLAS NUEVAS	TABLAS LEGADAS	Carga Inicial	Observaciones
NUE_PARAMETROS	LEG-TABLA-PARAM	Manual	La carga se realiza el día de la implantación
NUE_ARCHIVOS	LEG-ARCHIVOS	Automática	
NUE_MENSAJES	LEG-MENSAJES-1	Automática	Engloba los datos de dos tablas del sistema legado
	LEG-MENSAJES-2	Automática	
NUE_LOG	n/a	n/a	Sin correspondencia en sistema legado

Figura 20 – Equivalencia de tablas entre aplicación legada y nueva.

Fuente: elaboración propia



	NUE_PARAMETROS (DB2)				Correspondencia	LEG-TABLA-PARAM (ADABAS)			
	NOMBRE	TIPO	LONG	DESCRIPCIÓN		NOMBRE CAMPO	TIPO	LONG	DESCRIPCIÓN
PK	PAR-ID	CHAR	8	Identificación del parámetro	✓	TBL-TIPO	A	8	Nombre del parámetro
					✗	TBL-DESCR-CORTA	A	15	Resumen descripción
	PAR-DESCRIPCION	CHAR	50	Descripción del parámetro	✓	TBL-DESCR-LARGA	A	50	Descripción detallada del parámetro
	PAR-VALOR	CHAR	20	Contenido del parámetro	✓	TBL-VALOR	A	20	Contenido del parámetro
	PAR-FECHA-ALTA	DATE	60	Fecha de inicio de validez del parámetro	!	TBL-FECHA-FIN	N	10	Fecha de inicio de validez del parámetro
	PAR-FECHA-BAJA	DATE	40	Fecha de fin de validez del parámetro	!	TBL-FECHA-INI	N	10	Fecha de fin de validez del parámetro
	PAR-USUARIO-CTRL	CHAR	8	Usuario que modifica la tabla	✗				
	PAR-TMS-CTRL	TIMESTAMP	10	Fecha/hora de modificación de la tabla	✗				

Figura 21 – Equivalencia de columnas entre aplicación legada y nueva.

Fuente: elaboración propia

La columna “Correspondencia” de la Excel de equivalencia indica cómo es la relación entre los campos de acuerdo con la clave que se haya definido. Puede verse un ejemplo en la Figura 22.

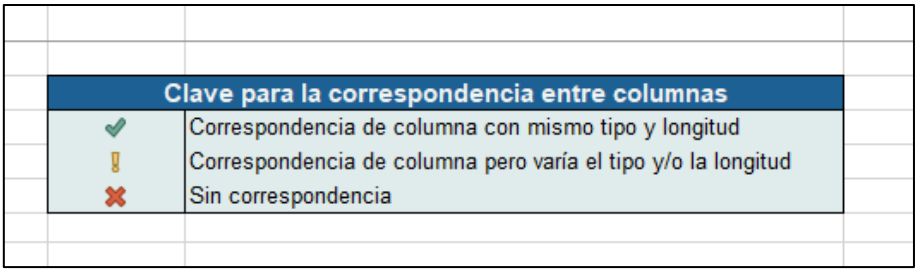


Figura 22 – Leyenda para interpretar la equivalencia entre campos de las BBDD legada y nueva.

Fuente: elaboración propia

Un problema similar se daba en lo relativo a las rutinas o los servicios que se desarrollan para exponer la información de la aplicación que se migra a aquellas otras con las que se relaciona. Dejar para las últimas etapas de desarrollo la construcción de este tipo de elementos hacía que las aplicaciones contaran con poco tiempo para la adaptación al nuevo sistema. Esto, además de aumentar el riesgo de que hubiera que retrasar la entrada en producción por no estar adaptada la totalidad de aplicaciones afectadas, aumentaba la posibilidad de que se produjeran errores por no poder contar con el tiempo de pruebas que hubiera sido deseable. Para reducir este factor de riesgo se decide considerar como subsistema independiente este conjunto de elementos y adelantar su desarrollo todo lo posible. Al igual que con el subsistema de BBDD, también aquí se genera un documento de equivalencias que además puede ser distribuido entre las aplicaciones afectadas para facilitarles el proceso de adaptación.

Por último, aunque de naturaleza diferente, hay otro conjunto de rutinas cuyo desarrollo temprano hace disminuir las posibilidades de desvío en fechas y horas de proyecto. Se trata de los elementos comunes, esto es, de rutinas que se llaman internamente desde diferentes puntos de la aplicación para (por ejemplo) escribir en un log, tratar errores, consultar una información concreta en una tabla, o llevar a cabo cálculos rutinarios como puede ser el de un dígito de control. A este último grupo también se le da entidad de subsistema para que pueda desarrollar de manera independiente.

5.4. FASE 4 – Estudio de la aplicación legada

Es difícil encontrar un proyecto que no contemple el estudio funcional de la aplicación legada en alguna de sus fases con independencia de la metodología que decida aplicar. Lo que ya no resulta tan sencillo de ver es que el estudio de esta aplicación legada se haga también desde un punto de vista técnico. En numerosas ocasiones las aplicaciones implementan los requisitos que les solicitan los usuarios aprovechando las posibilidades concretas que les da el entorno técnico en el que se desarrollan. Si este entorno cambia (como es el caso de una migración Mainframe) es posible que esa funcionalidad no pueda "calcarse" y a la hora de hacer la nueva implementación aparezcan problemas que obliguen a disminuir el ritmo de desarrollo para encontrar una solución acorde con el nuevo entorno de trabajo y que afecte lo menos posible al software que se ya se ha construido. Como ejemplo concreto de esta situación puede mencionarse el control del commit y el rollback. En entornos sin arquitectura es habitual que no haya restricciones en lo que respecta a la ejecución de estas acciones, pero en aquellos otros en los que sí hay arquitectura, probablemente sea esta última la que se encargue de controlarlas. Esto hace que deban rediseñarse los flujos de ejecución que consolidaban o descartaban cambios en BBDD antes de la finalización completa de una transacción.

5.5. FASE 5 – Estimación del esfuerzo, planificación del desarrollo completo y dimensionamiento del equipo.

Cuando se trata de aplicaciones grandes la estimación del esfuerzo no la realiza una sola persona, sino que son varios los analistas que participan de esta tarea. Para la valoración del número de horas necesarias para la migración de una aplicación es recomendable que existan una pautas generales o estándares que rijan esta tarea y dependa lo menos posible del criterio particular de las personas que llevan a cabo este cálculo. Para conseguir este fin, se propone trabajar con una herramienta de cálculo que, a partir de ciertos parámetros, determine la cantidad de horas necesarias para completar el desarrollo.

Los parámetros principales que se tienen en cuenta en la herramienta utilizada son el tipo de módulo que se quiere construir (con o sin patrón, proceso desatendido, mapa, etc.) y la dificultad que se le asigne a cada uno de ellos (Figura 23). Para un mayor ajuste, se ofrecen como parámetros secundarios el

número de llamadas a elementos de la propia aplicación (llamadas internas) y a elementos externos (llamadas a programas de otras aplicaciones) (Figura 24). La asignación de horas en función de todos los parámetros (principales y secundarios) se hace de acuerdo con unos baremos previamente establecidos (Figura 25) y calculados a partir del conocimiento aportado por los miembros con más experiencia del equipo. Estos valores pueden tener carácter general y ser utilizados por igual en varios proyectos, o particularizarse para alguno en concreto si presenta alguna característica peculiar. Por ejemplo, podría aumentarse el porcentaje dedicado a “Pruebas y Soporte Usuario” en el caso de que el número de usuarios responsables de probar la aplicación fuera elevado, o reducirse el de “Análisis y Cuadernos de Carga” si se contara en el proyecto con programadores experimentados.

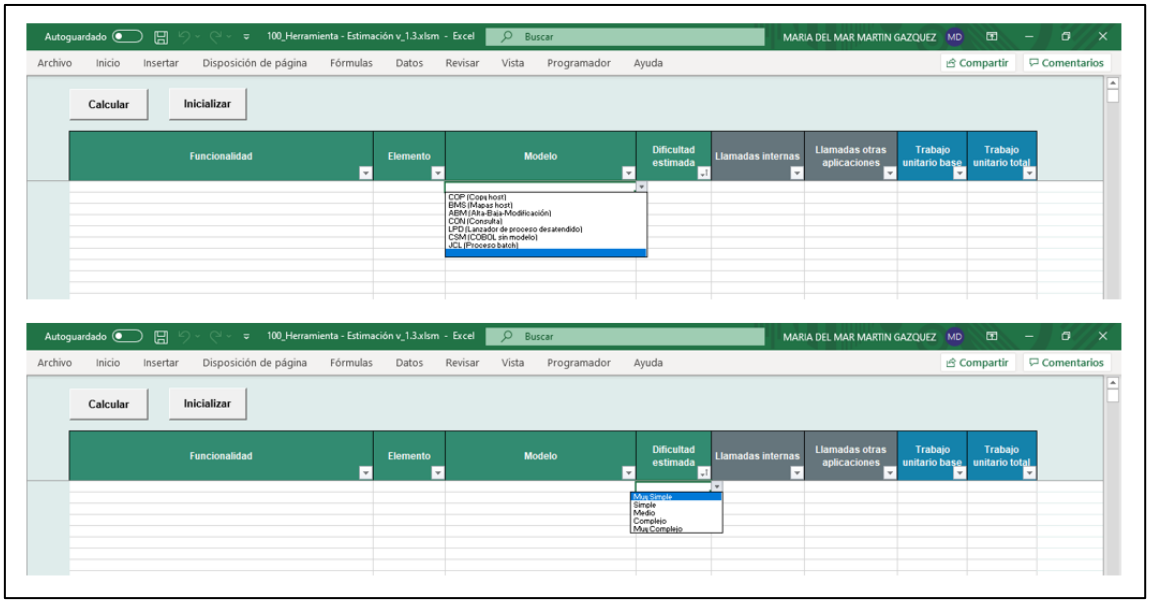


Figura 23 – Parámetros principales para el cálculo de la estimación.

Fuente: elaboración propia

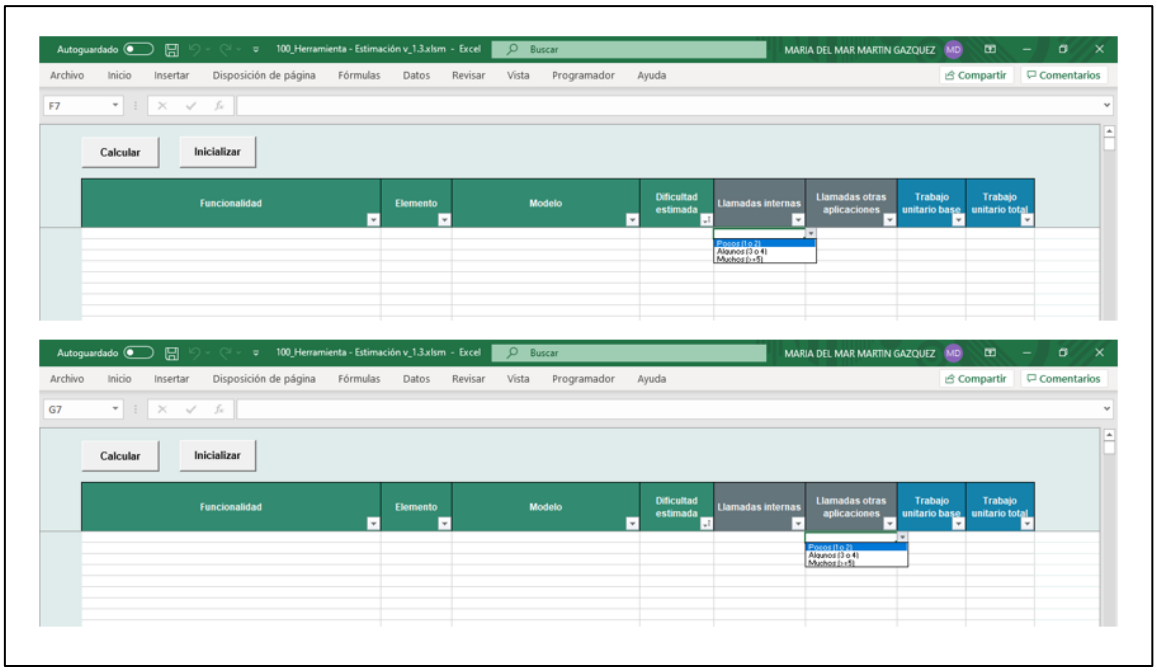
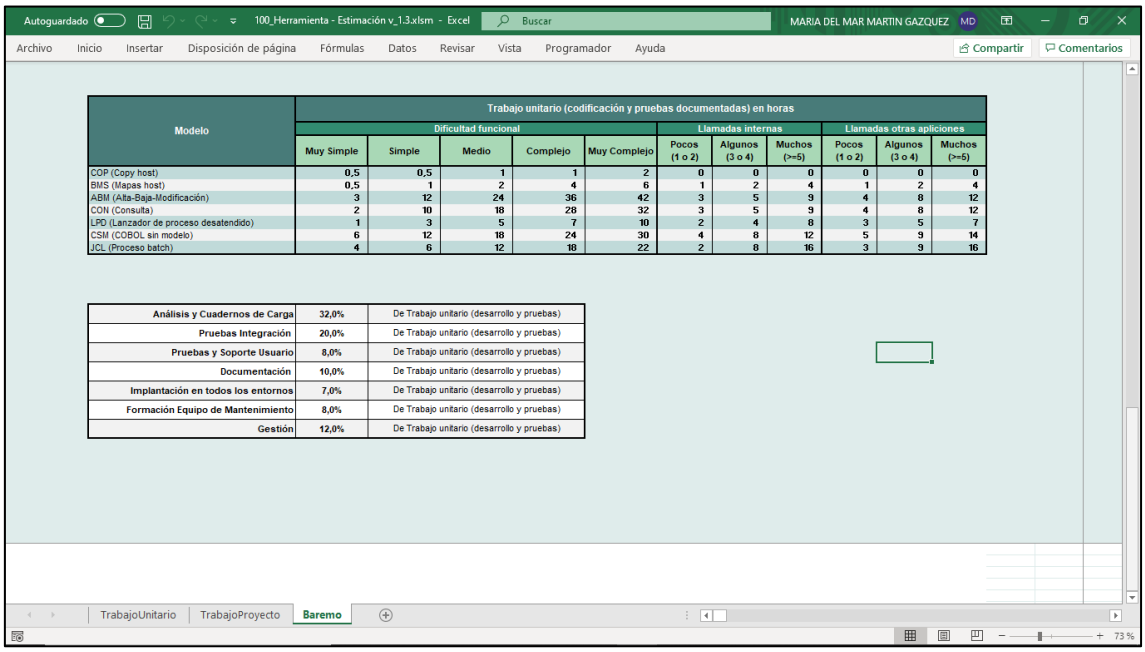


Figura 24 – Parámetros secundarios para el cálculo de la estimación.

Fuente: elaboración propia



Modelo	Trabajo unitario (codificación y pruebas documentadas) en horas										
	Dificultad funcional					Llamadas internas			Llamadas otras apliciones		
	Muy Simple	Simple	Medio	Complejo	Muy Complejo	Pocos (1 o 2)	Algunos (3 o 4)	Muchos (>=5)	Pocos (1 o 2)	Algunos (3 o 4)	Muchos (>=5)
COP (Copy host)	0,5	0,5	1	1	2	0	0	0	0	0	0
BMS (Mapas host)	0,5	1	2	4	6	1	2	4	1	2	4
ABM (Alta-Baja-Modificación)	3	12	24	36	42	3	5	9	4	8	12
CON (Consulta)	2	10	18	28	32	3	5	9	4	8	12
LPD (Lanzador de proceso desatendido)	1	3	5	7	10	2	4	8	3	5	7
CSM (COBOL sin modelo)	6	12	18	24	30	4	8	12	5	9	14
JCL (Proceso batch)	4	6	12	18	22	2	8	16	3	9	16

Análisis y Cuadernos de Carga	32,0%	De Trabajo unitario (desarrollo y pruebas)
Pruebas Integración	20,0%	De Trabajo unitario (desarrollo y pruebas)
Pruebas y Soporte Usuario	8,0%	De Trabajo unitario (desarrollo y pruebas)
Documentación	10,0%	De Trabajo unitario (desarrollo y pruebas)
Implantación en todos los entornos	7,0%	De Trabajo unitario (desarrollo y pruebas)
Formación Equipo de Mantenimiento	8,0%	De Trabajo unitario (desarrollo y pruebas)
Gestión	12,0%	De Trabajo unitario (desarrollo y pruebas)

Figura 25 – Baremos aplicados en el cálculo de la estimación.

Fuente: elaboración propia

Una vez rellenada la primera hoja de la herramienta de cálculo de estimaciones (Figura 26) se obtiene el esfuerzo total del proyecto (Figura 27). Como puede observarse, la herramienta permite agrupar por subsistemas, lo que facilita la planificación total del proyecto y la estimación de personas asignadas al mismo.

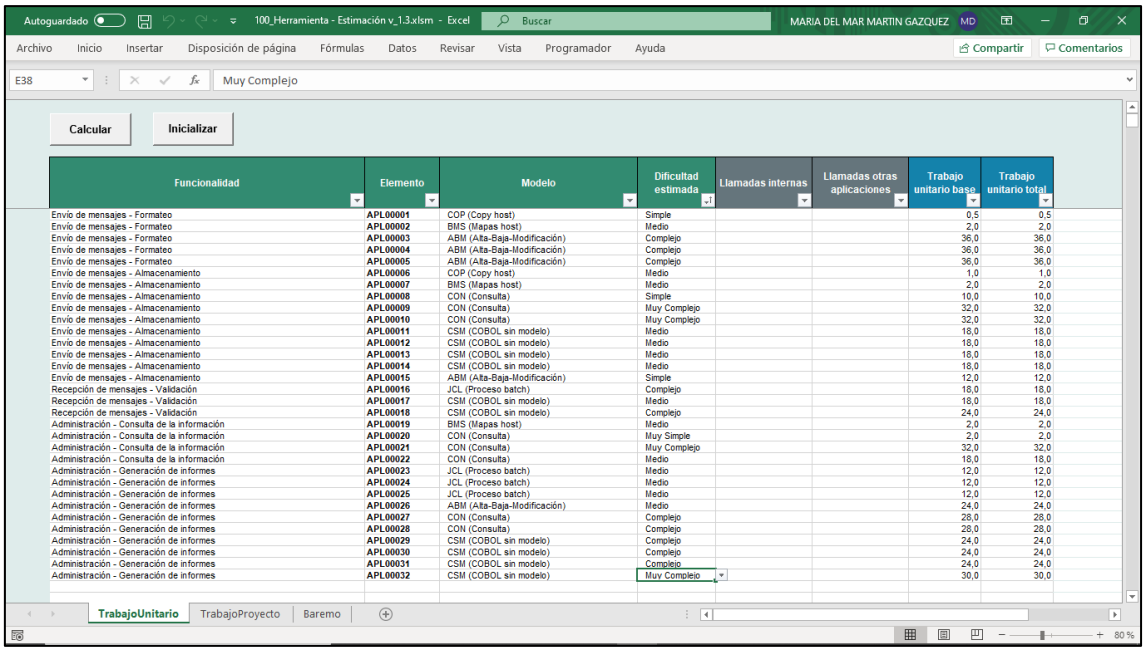


Figura 26 – Primera hoja de la herramienta de cálculo de la estimación. Ejemplo.

Fuente: elaboración propia

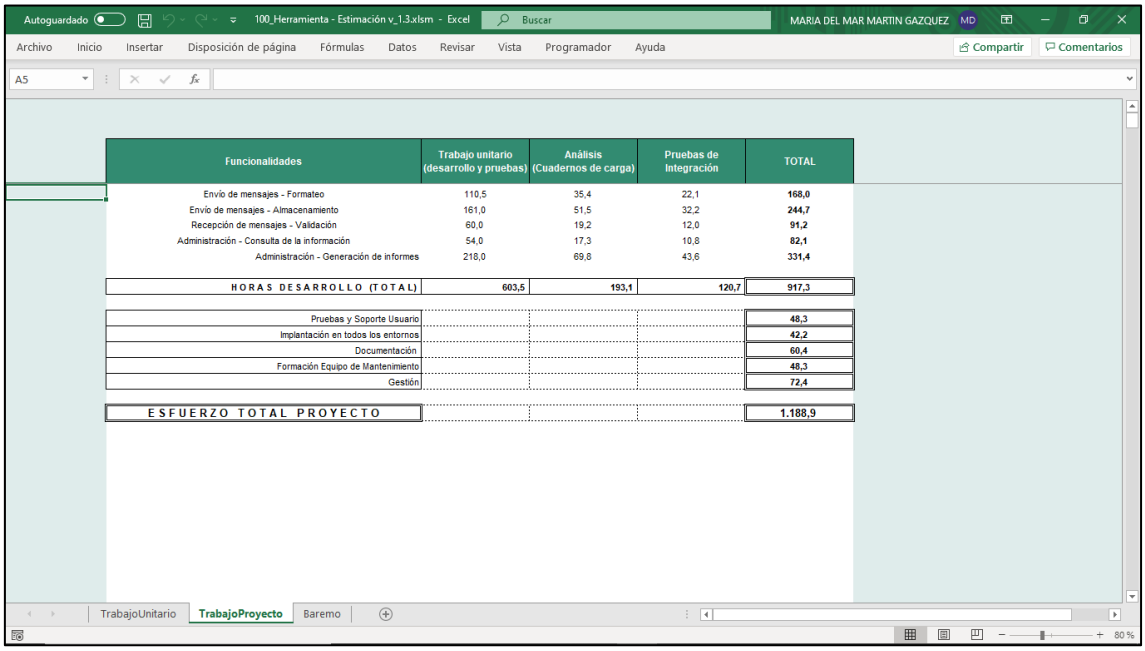


Figura 27 – Cálculo del esfuerzo total del proyecto.

Fuente: elaboración propia

Dependiendo de los requerimientos para la fecha de finalización se establece el número de líneas de trabajo paralelas en las que se quiere trabajar. Como norma general, cuanto más cercana sea esa fecha, más trabajo en paralelo será necesario realizar. Dentro de estas líneas de trabajo se irán construyendo los

distintos subsistemas. Como no todos son de la misma dimensión ni comienzan y terminan en las mismas fechas, es posible balancear la asignación de analistas y programadores, optimizando así el número de personas participantes en el proyecto. En la Figura 28 puede verse un ejemplo de distribución de subsistemas en distintas líneas de desarrollo.

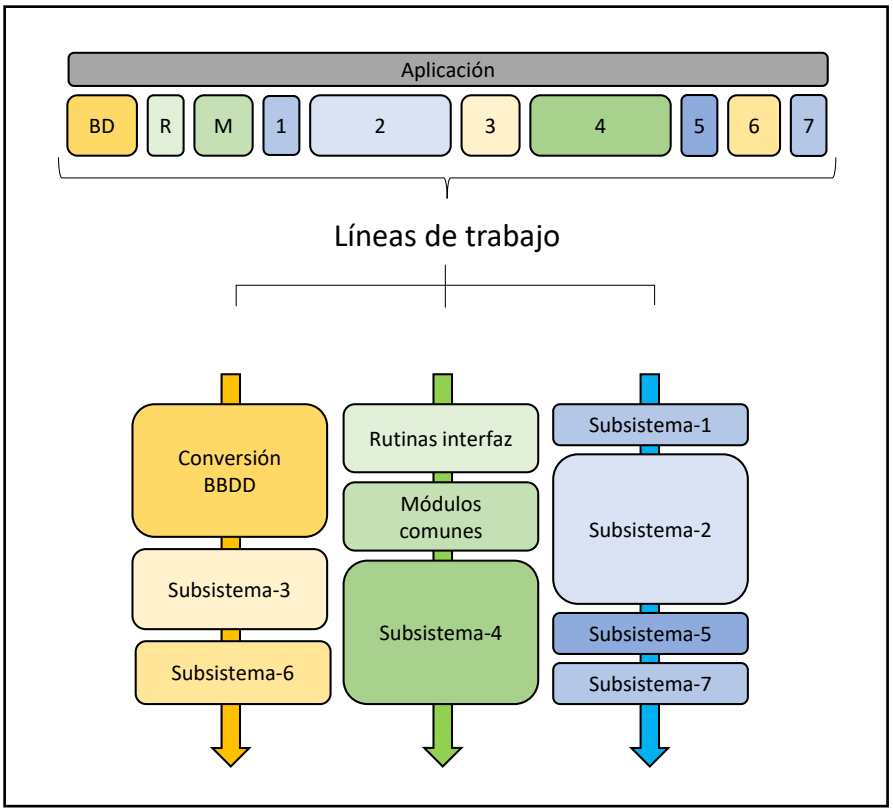


Figura 28 – Representación gráfica de la distribución de subsistemas en distintas líneas de trabajo

Fuente: elaboración propia.

La herramienta más utilizada para plasmar la planificación temporal del trabajo a realizar y de los componentes del equipo que colaborarán en cada subsistema y etapa de construcción es Microsoft Project. Debido a la larga duración que caracteriza a las migraciones Mainframe pueden surgir imperativos que justifiquen alguna modificación de la planificación inicial, pero ésta debe estar siempre presente porque es una referencia importante a la hora de cumplir los objetivos OnTime y OnBudget que se mencionaban en los capítulos iniciales.

5.6. FASE 6 – Proceso iterativo de construcción de subsistemas

Durante mucho tiempo ha sido costumbre en los proyectos Mainframe migrar las aplicaciones como entidades monolíticas. Es decir, el desarrollo abarcaba toda la funcionalidad que la aplicación ofrecía y el usuario final no dedicaba tiempo a sus pruebas hasta que no la tenía por completo a su disposición. Esta manera de trabajar presenta ciertos inconvenientes. Uno de ellos es que los posibles retrasos en codificación y pruebas en cualquier parte de la aplicación repercuten necesariamente en la fecha de inicio de pruebas de usuario. También puede suceder que cuando el usuario comience las pruebas de una determinada funcionalidad haya pasado un largo período de tiempo desde que se terminó de codificar y esto haga más difícil localizar a qué se deben los problemas reportados en las incidencias. También añade dificultad en la resolución de incidencias el hecho de que al final de los proyectos, no habiendo ya tareas de desarrollo propiamente dichas, el número de programadores se reduzca drásticamente y el pequeño grupo que se mantenga deba hacer frente a cualquier error que haya que solucionar, conozca o no el código que debe ser modificado. Todas estas circunstancias repercuten negativamente en la finalización del proyecto en la fecha acordada y dentro del presupuesto asignado. Para evitarlo, la construcción de la aplicación se lleva adelante atendiendo a la división en subsistemas que se hizo en la FASE-3 y el usuario va recibiendo la aplicación en entregas parciales. Para llevar el control de qué subsistemas se pueden entregar al usuario porque ya cumplen con todos los requisitos definidos para ellos y no tienen incidencias o mejoras pendientes de implementar, puede utilizarse la herramienta Silk Central. Esta herramienta de Microfocus facilita el registro de los requerimientos funcionales y no funcionales de los subsistemas, así como la gestión y seguimiento de los casos de pruebas que deben ejecutarse para garantizar una correcta implementación del software (Figura 29).

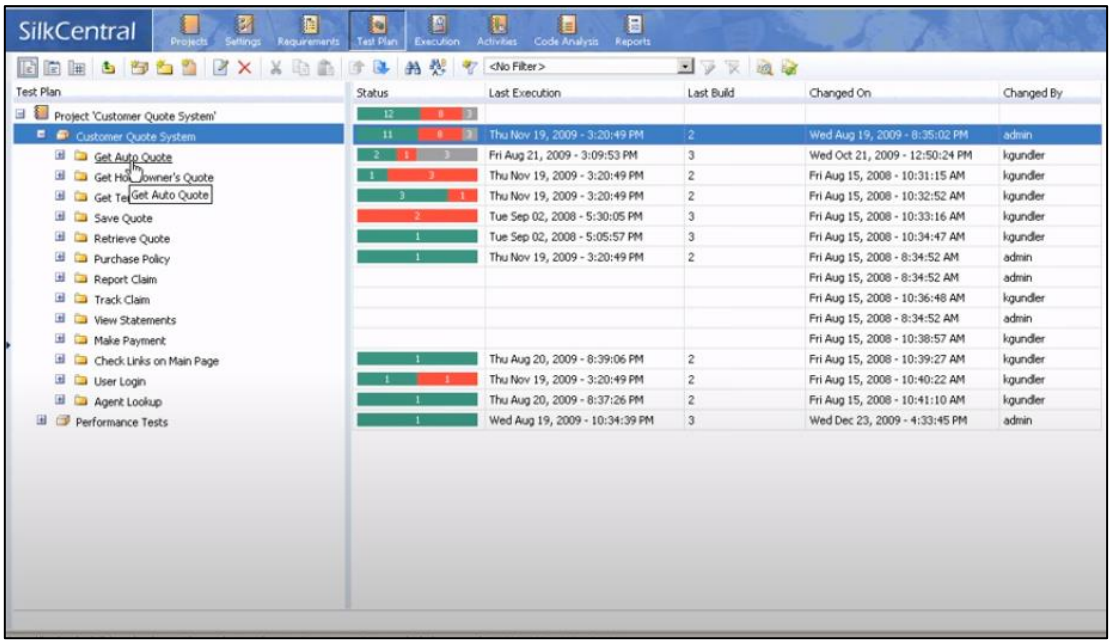


Figura 29 – Silk Central. Consulta de cobertura y avance de las pruebas.

Fuente: TalaBaySystems[23]

Entre las ventajas que ofrece este nuevo modo de trabajar se encuentran la participación e implicación del usuario en el proyecto desde una etapa temprana, una mayor rapidez en la resolución de incidencias debido a la cercanía temporal con el desarrollo del software que se prueba y una gran posibilidad de que quien debe resolver el problema reportado haya participado en la codificación de la funcionalidad que se está probando. Otra de las grandes ventajas de abordar el desarrollo por subsistemas es la oportunidad de poder hacerlo de manera paralela, es decir, pueden definirse subequipos que trabajen de manera independiente en cada uno de los subsistemas Esto hace que disminuya el riesgo de que se desplace la fecha de fin de proyecto. Si una de las ramas de desarrollo sufre un inconveniente y se retrasa, el resto puede seguir avanzando sin problemas. Puede que una parte le llegue más tarde al usuario, pero la repercusión sobre su planificación de pruebas, así como la global del proyecto, no se vea afectada o lo haga de manera más reducida. También da pie a que ante un cuello de botella en un subequipo éste pueda reforzarse de manera interna con la reasignación de alguno de los componentes de los otros subequipos y no sea necesario buscar apoyo fuera del ámbito del proyecto.

[23] https://www.youtube.com/watch?v=tAFHfUwDdbM (22-05-2021)

Otro aspecto importante de este trabajo en paralelo de varios subsistemas y repartido en subequipos es la posibilidad de dotar a estos últimos de cierta autonomía e incluir prácticas Agile en su gestión: una vez hechos los equipos, asignado el trabajo y definida una fecha de entrega, pueden ser los propios componentes del equipo los que decidan quién hace qué y lleven el control principal del avance y de lo que queda por construir. La labor del jefe de proyecto en este aspecto es la de supervisar el trabajo de los equipos para garantizar que todos ellos funcionan correctamente y, en caso de ser necesario, intervenir para dar solución a problemas similares al que se comentaba anteriormente sobre cómo resolver posibles cuellos de botella.

5.7. FASE 7 – Gestión de cambios

El solapamiento entre el desarrollo del nuevo software y la adaptación de éste a las modificaciones realizadas en la aplicación origen puede comenzar el mismo día en que se inicia el proyecto. Esta concurrencia de nuevo desarrollo con modificaciones sobre el software ya construido se acentúa cuando comienzan las entregas parciales al usuario, ya que, a partir de ese momento, se suma también la resolución de incidencias. Llegados a este punto, se suele observar una acumulación de trabajo que, por lo general, acaba repercutiendo en la planificación inicial del proyecto y poniendo en peligro las fechas de entrega acordadas en las etapas iniciales. Otra observación importante es que el problema no es únicamente el aumento casi repentino del volumen de trabajo, sino la descoordinación entre las tareas que van abordando el equipo de trabajo y los responsables de la aplicación o los usuarios finales. Por ejemplo, en ocasiones se resuelven incidencias que los usuarios no verifican hasta varias semanas más tarde, o se dedica tiempo a repercutir modificaciones de la aplicación legada en un subsistema que el usuario no tiene previsto revisar a corto plazo. Y sucede también al contrario: incidencias o adaptaciones que pueden parecer no prioritarias resultan ser las más urgentes para el usuario y se resuelven o implementan demasiado tarde.

De estas situaciones se desprende que el problema que subyace es el de una priorización de tareas inadecuada. La planificación que los usuarios hagan de sus pruebas deja de ser transparente para el equipo de desarrollo y se hace necesario consensuar cuándo se aborda qué para el caso de las incidencias y

de la adaptación a modificaciones hechas en la aplicación original. Esta coordinación se consigue con una correcta gestión de cambios y, como ya se expuso en el capítulo dedicado al desarrollo de la FASE-7, el modelo ITIL aporta las etapas y los roles que pueden ayudar a llevar al proyecto a buen término. El registro de todas las tareas adicionales al desarrollo en sí de la aplicación, las reuniones entre el jefe de proyecto, el responsable funcional y los usuarios a modo de Comité de Cambios, la planificación consensuada de la implementación de estos cambios, así como de los períodos de pruebas, permite atender las demandas que llegan al equipo de desarrollo dentro del plazo establecido para la finalización global del proyecto.

Para la implementación de este proceso de gestión de cambios se puede recurrir a herramientas de gestión que facilitan el registro de modificaciones e incidencias y la coordinación entre las distintas partes interesadas en el proyecto. A modo de ejemplo pueden mencionarse JIRA, de Altassian.

El modelo ITIL contempla la existencia de una CMDB para el registro y seguimiento de las solicitudes de cambio y JIRA es un producto que puede servir para este fin. En él se registran las incidencias y mejoras que se quieren resolver o repercutir en el software ya construido (Figura 30)

Después las tareas pueden priorizarse y gestionarse de manera sencilla a través de pizarras en las que se distribuyen en función del estado en el que se encuentren o de la prioridad que les haya sido asignada (Figura 31).

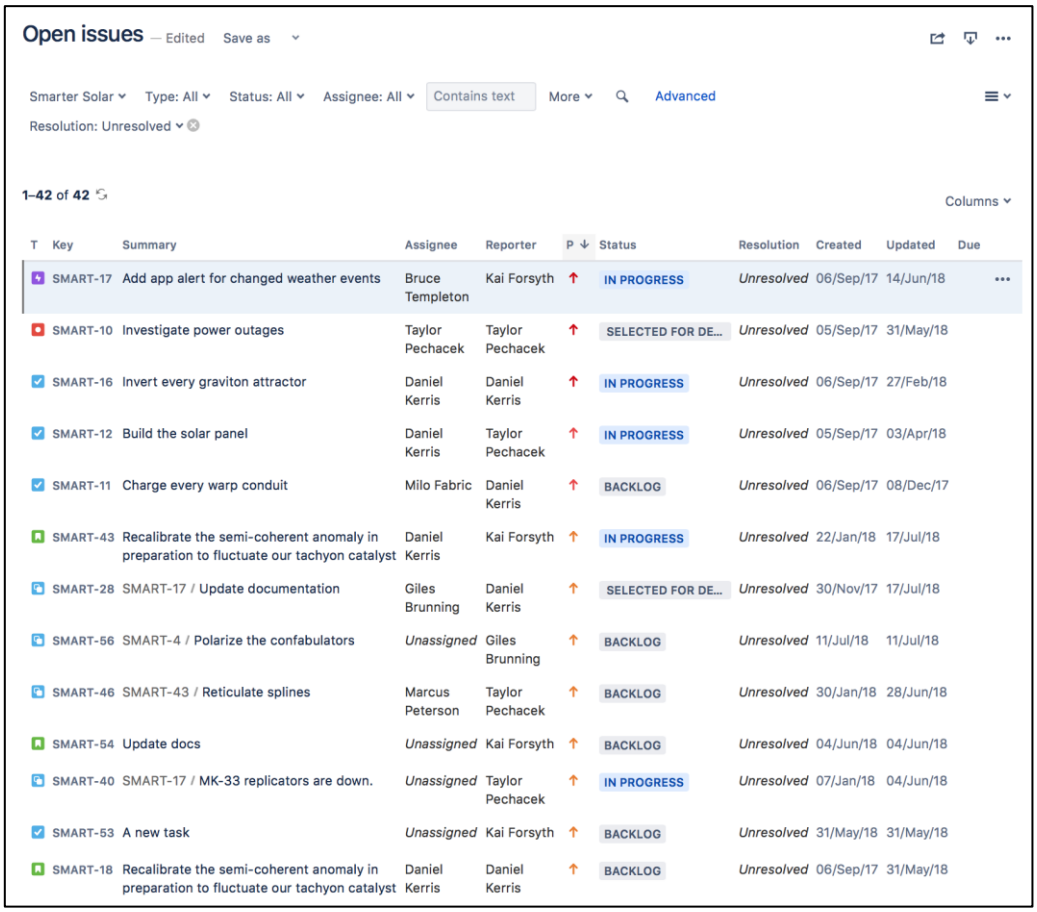


Figura 30 – JIRA. Relación de tareas registradas

Fuente: Altassian[24]

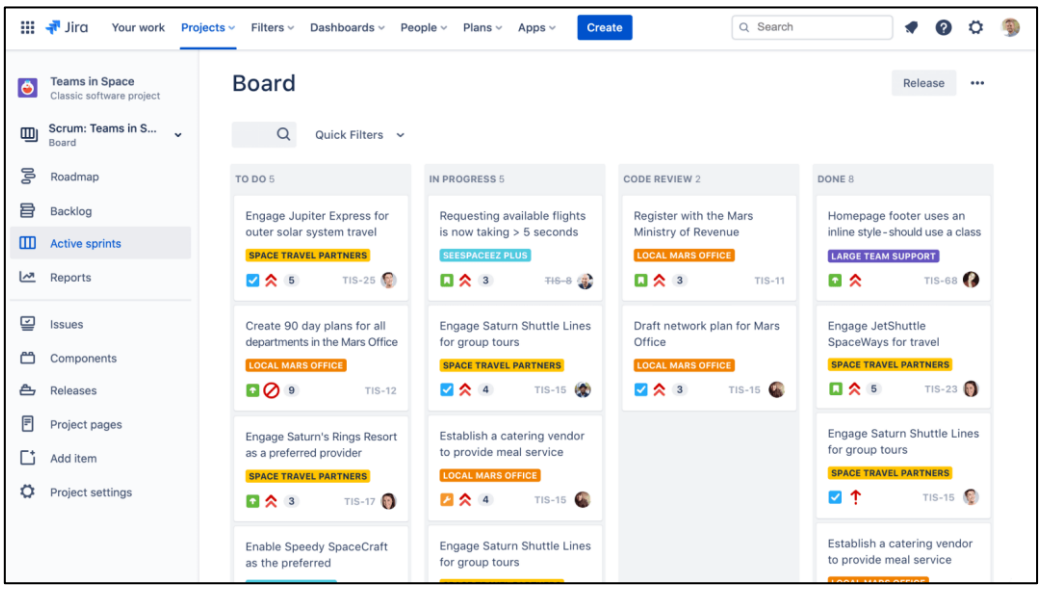


Figura 31 – JIRA. Pizarra con las tareas registradas

Fuente: Altassian[25]

[24] https://support.atlassian.com/jira-cloud-administration/docs/configure-the-default-issue-navigator/ (22-05-2021)

[25] https://www.atlassian.com/es/software/jira (22-05-2021)

Estas pizarras son el documento de trabajo de las reuniones del Comité de Cambios en las que se decide la prioridad y orden de resolución de todas las tareas que entran dentro de la gestión de cambios. El camino que una vez aprobada sigue una determinada tarea queda definido también en JIRA (Figura 32). Este flujo es particularizable, por lo que puede adaptarse a las necesidades propias de cada proyecto.

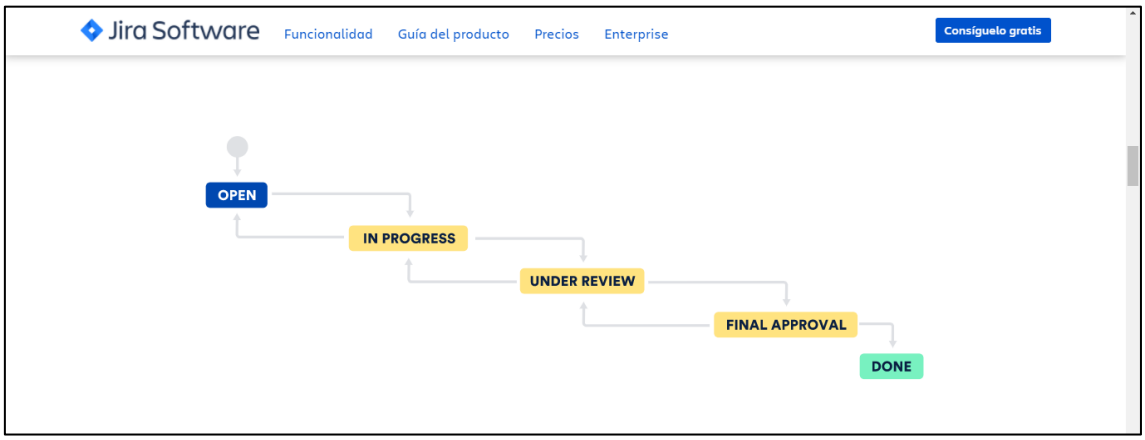


Figura 32 – JIRA. Flujo de trabajo.

Fuente: Altassian[26]

Cuando se intenta combatir el inconveniente del aumento del volumen de trabajo sin tener en cuenta este punto de vista más organizativo y se toman únicamente medidas como añadir nuevos miembros al equipo, o que el equipo ya asignado realice un sobreesfuerzo de manera continua, pocas veces se resuelve este problema conocido, esperado y (en apariencia) inevitable. La primera alternativa puede aportar como ventaja aparente que, si el equipo aumenta, el trabajo a realizar por cada uno de los miembros será menor. Sin embargo, las personas que llevan más tiempo en el proyecto deben dedicarse a enseñar a los recién llegados, y estos a su vez necesitan un cierto periodo de adaptación para comprender el funcionamiento de la aplicación y el entorno de trabajo en el que se desarrolla. La segunda alternativa que se menciona (el sobreesfuerzo) solo es efectiva cuando se aplica en momentos muy determinados del proyecto y durante un tiempo limitado.

[26] https://www.atlassian.com/es/software/jira (22-05-2021)

5.8. FASE 8 – Elaboración del plan de implantación y puesta en producción

Implantar una aplicación de gran tamaño en poco tiempo es un reto al que deben enfrentarse los proyectos de migración Mainframe y aquellos que descuidan la elaboración del plan de implantación tienen un mayor número de incidencias tras la entrada en producción. Por muy bien que haya ido un proyecto, los problemas debidos a no tener una relación completa de las tareas de definición de la infraestructura, de la solicitud de permisos, de la documentación necesaria para la entrega del proyecto, o de los elementos software que componen la aplicación, pueden enmascarar todo el buen trabajo que se haya hecho anteriormente. Problemas concretos que aparecen cuando no se ha hecho un buen plan de implantación son: mal funcionamiento de las comunicaciones por no haber activado el canal correspondiente, usuarios sin poder acceder a la aplicación por no haberles sido asignados los permisos y roles necesarios para acceder al menú de la nueva aplicación, trabajos críticos no ejecutados por no haber hecho llegar a los responsables de la planificación el documento con las condiciones de arranque de los procesos o errores de ejecución por no estar compilado algún programa en el entorno de producción.

Al igual que en la FASE-1 se propone trabajar con un formulario que sirva de guía y ayude a tener presente todos los puntos importantes para la elaboración (en este caso) del plan de implantación (Figura 29). Una vez cumplimentado, al pulsar el botón ACEPTAR se traslada la información a una hoja Excel preparada para imprimir el informe en formato .pdf (Figura 30). El resultado de este informe ya impreso puede verse en la Figura 31.

Este formulario puede ser creado por personal experto en la dirección de proyectos y servir de guía al jefe de proyecto de la migración. De esta manera, incluso tratándose de una persona con poca experiencia en este rol, en el plan de implantación no faltarán las tareas esenciales que garantizan el éxito en esta última etapa del proyecto.

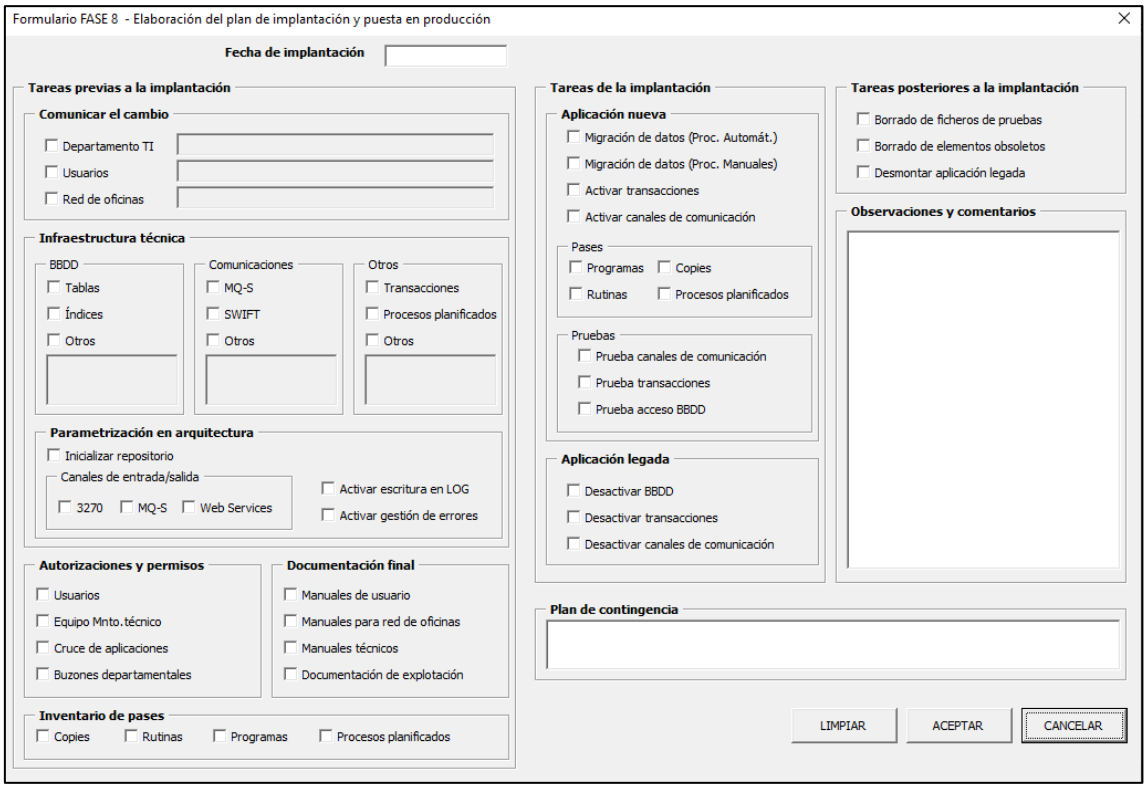


Figura 33 – Ejemplo de formulario para la creación del plan de implantación.

Fuente: elaboración propia

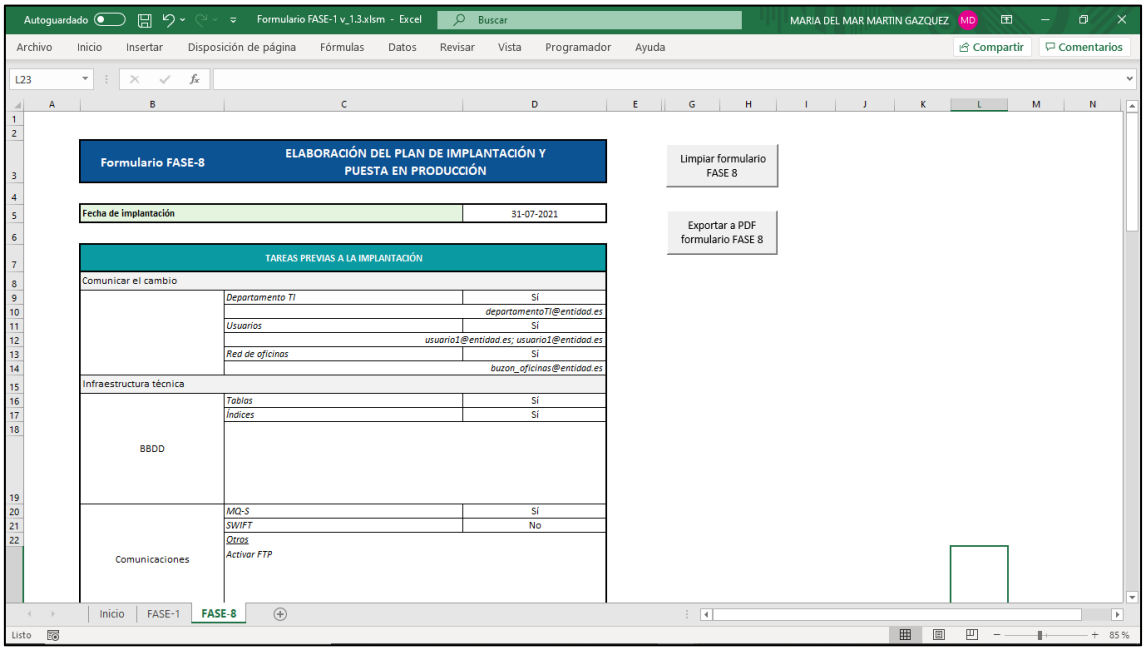
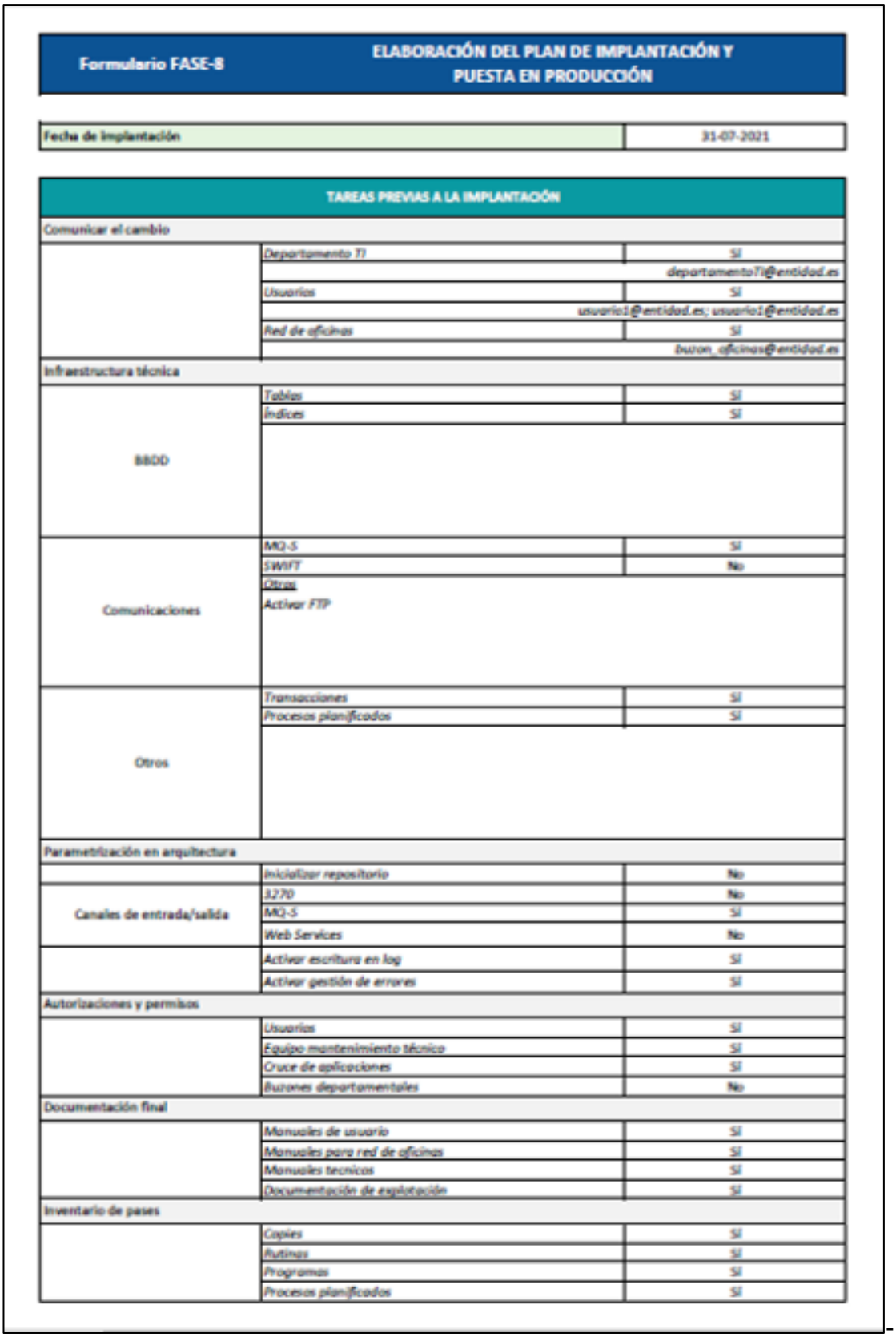


Figura 34 – Información recogida en el formulario de FASE-8 y trasladada a una hoja Excel.

Fuente: elaboración propia



Formulario FASE-8	ELABORACIÓN DEL PLAN DE IMPLANTACIÓN Y PUESTA EN PRODUCCIÓN	
Fecha de implantación		31-07-2021
TAREAS PREVIAS A LA IMPLANTACIÓN		
Comunicar el cambio		
	Departamento TI	Sí
		departamentoTI@entidad.es
	Usuarios	Sí
		usuario1@entidad.es; usuario1@entidad.es
	Red de oficinas	Sí
		buzon_oficinas@entidad.es
Infraestructura técnica		
BBDD	Tablas	Sí
	Índices	Sí
Comunicaciones	MQ-S	Sí
	SWIFT	No
	Otros Activar FTP	
Otros	Transacciones	Sí
	Procesos planificados	Sí
Parametrización en arquitectura		
Canales de entrada/salida	Inicializar repositorio	No
	3270	No
	MQ-S	Sí
	Web Services	No
	Activar escritura en log	Sí
	Activar gestión de errores	Sí
Autorizaciones y permisos		
	Usuarios	Sí
	Equipo mantenimiento técnico	Sí
	Cruce de aplicaciones	Sí
	Buzones departamentales	No
Documentación final		
	Manuales de usuario	Sí
	Manuales para red de oficinas	Sí
	Manuales tecnicos	Sí
	Documentación de explotación	Sí
Inventario de pases		
	Copies	Sí
	Rutinas	Sí
	Programas	Sí
	Procesos planificados	Sí

Figura 35 – Informe del plan de implantación (tareas previas) obtenido a partir del formulario de FASE-8.

Fuente: elaboración propia

6. CONCLUSIONES Y LÍNEAS FUTURAS

6. Conclusiones y líneas futuras

Como final del proceso de investigación y diseño, se exponen las conclusiones que pueden extraerse de este trabajo y las posibles líneas futuras de investigación.

6.1. Conclusiones

Las conclusiones relacionan los objetivos definidos al inicio del proyecto con los resultados obtenidos al final del mismo. El punto de partida de este recorrido han sido las condiciones que caracterizan a los proyectos de migración Mainframe dentro del ámbito bancario, así como los datos publicados acerca de los factores de riesgo que, objetivamente, afectan al éxito de los proyectos de software. Todo ello reforzaba la idea de que las migraciones Mainframe tenían poca probabilidad de éxito y que esto no fuera así, iba a depender en gran parte de la aportación que el jefe de proyecto pudiera hacer en la organización, dirección y puesta en marcha del proyecto. El objetivo principal y punto final del TFM era encontrar un modelo de trabajo que evitara esta circunstancia cubriendo una serie de objetivos secundarios:

1. *Estudio de las metodologías de migración ya existentes (específicas o no de entornos Mainframe).*

El modelo de trabajo que se expone en este TFM tiene su origen en la experiencia real y se ha puesto en práctica en situaciones reales con resultados satisfactorios. Sin embargo, el desconocimiento de que algunas de las ideas puestas en marcha estuvieran avaladas por publicaciones científicas, hacían dudar de si estos buenos resultados se debían a la casualidad o a haber tomado la decisión correcta en el momento adecuado. El estudio realizado ha permitido consolidar las buenas prácticas utilizadas en los proyectos e incluirlas como parte fundamental del modelo de trabajo que se propone. Por citar un ejemplo puede mencionarse la conversión de la base de datos, que (como ya ocurre en alguna de las metodologías estudiadas) se considera una tarea principal y adquiere entidad de subsistema, pasando a ser uno de los pilares del proceso de migración.

2. *Seleccionar y analizar estándares, ciclos de vida y modelos de gestión aplicables al ámbito bancario Mainframe.*

Habitualmente se da por hecho que en el entorno Mainframe solo se pueden aplicar modelos organizativos tradicionales y que no hay cabida para otro tipo de metodologías debido, principalmente, al carácter monolítico y al gran volumen de sus aplicaciones. El estudio de estándares propios y ajenos al mundo Mainframe (ITIL, SCRUM, etc.) ha permitido ampliar esta perspectiva, proponiendo (por ejemplo) una organización de equipos más flexible y con un cierto grado de autogestión, algo que hasta ahora había sido prácticamente impensable.

3. *Determinar las fases y tareas de las que se compone el modelo de trabajo para un proyecto de migración de una aplicación bancaria.*

A partir del estudio e investigación de metodologías de migración y estándares de gestión y gracias a la experiencia conseguida por la participación en proyectos de migración Mainframe, se han definido las 8 fases de las que consta el nuevo modelo de trabajo:

- FASE 1 – Estudio del entorno y la tecnología destino.
- FASE 2 – Determinación del alcance funcional de la migración.
- FASE 3 – Definición de subsistemas dentro de la aplicación.
- FASE 4 – Estudio de la aplicación legada.
- FASE 5 – Estimación del esfuerzo, planificación del desarrollo completo y dimensionamiento del equipo.
- FASE 6 – Proceso iterativo de construcción de subsistemas.
- FASE 7 – Gestión de cambios.
- FASE 8 – Elaboración del plan de implantación y puesta en producción.

Tan importante como definir el objetivo particular de cada una de estas fases, es determinar el orden en que deben ser aplicadas. Tener que escribir de manera pormenorizada las tareas que se ejecutan dentro de cada una ha permitido establecer con precisión las dependencias entre ellas y, en consecuencia, asignar el orden más propicio para la consecución de resultados satisfactorios en los proyectos de migración Mainframe

4. *Presentar un caso de estudio en el que se aplique el modelo de trabajo propuesto haciendo uso de herramientas tanto de creación propia como de uso general.*

El trabajo relacionado con este último objetivo ha permitido mostrar a qué problemas en particular se da solución en cada una de las fases y cómo el uso de herramientas facilita su resolución. Sin los ejemplos concretos que se describen en el capítulo dedicado al caso de uso la propuesta del nuevo modelo de trabajo habría quedado en un plano excesivamente teórico, muy alejado de su aplicación práctica.
Sugerir el uso de herramientas específicas en cada fase allana el camino para aquellos que consideren oportuno aplicar esta nueva propuesta en sus proyectos de migración y facilita la comprensión de cuál es el objetivo que se pretende cubrir en cada una de las fases.

Como conclusión general puede decirse que este TFM ha permitido consolidar, dar forma, y dotar de respaldo teórico a un modelo de trabajo surgido de la observación y resolución de problemas en proyectos de migración Mainframe.

6.2. Líneas futuras

Se aprovecha este capítulo de líneas futuras para relacionar el proyecto con las asignaturas cursadas durante el máster.

6.2.1. Generación automática de código (3110501)

Los proyectos de migración son un buen escenario de trabajo para las herramientas de generación de código. Si bien el uso de herramientas generales puede no ser todo lo productivo que cupiera esperar, los generadores de código particulares o particularizados sí pueden ofrecer un buen servicio en este tipo de desarrollos.
Una posible línea de trabajo futuro podría ser el diseño y desarrollo de una herramienta de generación de código que, a partir de las especificaciones de las tablas origen y destino, construyera los scripts de transformación de datos entre ambas BBDD. Una segunda línea de trabajo dentro de este mismo ámbito de actuación sería la utilización de un generador de código para la elaboración de documentación.

6.2.2. Especificación de los sistemas software (31105024)

Los conocimientos adquiridos en esta asignatura podrían aplicarse en el desarrollo de la línea de trabajo descrita en el punto anterior, estableciendo de manera formal las especificaciones de los generadores de códigos propuestos. Al tratarse de un TFM englobado en el ámbito de la gestión y mejora de los procesos de software no resulta sencillo encontrar una línea de trabajo independiente para este campo de la ingeniería del software.

6.2.3. Arquitecturas Orientadas a Servicios (31105058)

Aunque este trabajo se ha centrado en las migraciones de proyectos con origen y destino en Mainframe, la tendencia aparecida en los últimos años de virtualizar software (SaaS), plataformas (PaaS), infraestructuras (IaaS) e incluso hardware (HaaS) y llevarlos a La Nube está alcanzando también al entorno z/OS. Una posible línea futura de investigación sería la de estudiar en qué medida y en qué condiciones el modelo de trabajo propuesto sería aplicable a proyectos de virtualización del Mainframe en un entorno de arquitectura orientada a servicios (SOA). Como puntos importantes que deberían ser tenidos en cuenta estarían:

- La seguridad en el acceso al sistema y en la transmisión de información.
- Los niveles de interoperabilidad: técnico, semántico y organizativo.
- El uso de ESB (Enterprise Service Bus) para la virtualización de recursos.
- La integración con otros sistemas independientemente de la tecnología en la que estuvieran desarrollados, prestando especial atención a los tiempos de respuesta.
- La utilización de software libre, que evitaría o reduciría el pago de elevadas licencias de software.

6.2.4. Gestión y mejora de los procesos software (31105062)

Todo el TFM está enmarcado dentro del ámbito de esta asignatura, pero eso no impide que puedan plantearse futuras líneas de trabajo que amplíen o extiendan su contenido.

En algunas de las fases de la metodología propuesta se aplican conceptos de otras ya existentes tradicionales y ágiles, y de modelos de buenas prácticas aplicadas a los entornos TI (ITIL). Teniendo en cuenta que estas metodologías y

buenas prácticas evolucionan con el tiempo, una posible línea de investigación podría consistir en adaptar el modelo de trabajo propuesto a las nuevas versiones de SCRUM, ITIL, etc. que puedan ir apareciendo a lo largo del tiempo. Otra línea de trabajo podría estar en la ampliación del modelo metodológico propuesto en el TFM y abarcar la estandarización en el uso de las herramientas software que se han referenciado (JIRA, Silk Central, etc.). Por ejemplo, estableciendo la manera de registrar y agrupar incidencias, requisitos o casos de prueba.

6.2.5. Sistemas difusos para toma de decisiones (31105081)

Tomando como ejemplos la FASE 5 (Estimación del esfuerzo, planificación del desarrollo completo y dimensionamiento del equipo) y la FASE 6 (Proceso iterativo de construcción de subsistemas), podrían incluirse en el modelo de trabajo propuesto sistemas de ayuda a la toma de decisiones basados en la lógica difusa para establecer (por ejemplo) el orden en el que deben implementarse los distintos subsistemas, la composición del equipo, hacer seguimiento del desarrollo, etc. Esta línea de trabajo estaría orientada por tanto a determinar en qué fases de la metodología podrían aplicarse los sistemas difusos para toma de decisiones, seleccionar para cada caso el más adecuado e implementar ejemplos concretos.

7. BIBLIOGRAFÍA

7. Bibliografía

ALTHANI, B. AND KHADDAJ, S., "Systematic Review of Legacy System Migration," *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, Anyang, pp. 154-157, 2017.

BENNETT, K., "Legacy systems: coping with success," in *IEEE Software*, vol. 12, no. 1, pp. 19-23, 1995.

Bergmayr, A., Bruneliere, H., Canovas Izquierdo, J., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., et al., "Migrating legacy software to the cloud with artist, " *17th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 465–468, 2013

BISBAL, J., LAWLESS, D., WU, B., GRIMSON, J., "Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues.", *IEEE Software 16*, pp. 103-111, 1999.

BRITO, K. D. S., GARCÍA, V. C., and Meira, S. R. d. L., "Experiences from a Brazilian Bank Reengineering Project," *2010 14th European Conference on Software Maintenance and Reengineering*, Madrid, pp. 66-67, 2010.

BRODIE, M. AND STONEBRAKER, M., "DARWIN: On the Incremental Migration of Legacy Information Systems," *Technical Report No. TR-0222-10-92-165, GTE Laboratories Inc.*, Waltham, Massachusetts, 1993.

BOOK, M., GRAPENTHIN, S. AND GRUHN, V., "Risk-aware Migration of Legacy Data Structures," *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, pp. 53-56, 2013.

CHARETTE, R.N., "No one notices the creaky software systems that run the world - until they fail," in IEEE SPECTRUM septiembre 2020, pp. 24-30.

DE LUCIA, A., "Developing Legacy System Migration Methods and Tools for Technology Transfer", *Software Practice Experience*, vol. 38, no. 13, pp. 1333-1364, 2008.

DORNINGER, B., MOSER, M. AND PICHLER, J., "Multi-language re-documentation to support a COBOL to Java migration project," *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Klagenfurt, pp. 536-540, 2017.

FORITE, L. AND HUG, C., "FASMM: Fast and Accessible Software Migration Method," *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, Marrakech, pp. 1-12, 2014.

FUENTES-FERNÁNDEZ, R., PAVÓN, J. AND GARIJO, F., "A model-driven process for the modernization of component-based systems," *Sci. Comput. Program.,* pp. 247–269, 2012.

GÓMEZ PALOMO, S.R. AND MORALEDA GIL, E., "Aproximación a La Ingenieria Del Software," Madrid: Ramón Areces, 2020, 2ª ed. (ISBN: 9788499613291).

HASSELBRING, W., FUHR, A. AND RIEDIGER, V., "First International Workshop on Model-Driven Software Migration (MDSM 2011)," *2011 15th European Conference on Software Maintenance and Reengineering*, Oldenburg, pp. 299-300, 2011.

MARTENS, A., BOOK, M. AND GRUHN, V., "A data decomposition method for stepwise migration of complex legacy data," *Software-Practice & Experience*, vol. 49, no. 2, pp. 1-19, 2019.

MENYCHTAS, A, SANTZARIDOU, C., KOUSIOURIS, G., VARVARIGOU, T., Orue-Echevarria, L., Alonso, J., et al., "ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud," *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, pp. 424-431, 2013.

MOMOCS, Model driven Modernisation of Complex Systems, DELIVERABLE D31, METHODOLOGY SPECIFICATION. Disponible (Enero 2021): https://cordis.europa.eu/docs/projects/cnect/6/034466/080/deliverables/D31-final.pdf

MOMOCS, Model driven Modernisation of Complex Systems, DELIVERABLE D32, METHODOLOGY STANDARDS. Disponible (Enero 2021): https://cordis.europa.eu/docs/projects/cnect/6/034466/080/deliverables/D32-final.pdf

RÍOS HUÉRCANO, S., "ITIL v3 Manual íntegro" Biable Management, Excellence and Innovation. Sevilla. Disponible (Mayo 2021) https://es.scribd.com/document/332172572/Manual-ITIL-v3-Integro-pdf

ROYCE, W. W., "Managing the Development of Large Software Systems," Proceedings IEEE WESCON, Los Angeles, pp. 1-9, 1970.

SILVA, D., ORTIZ GALVÁN, J., ANDRADE, H. AND RIVERA-LOPEZ, R., "Una Propuesta de Metodología para la Migración de Sistemas Heredados," pp. 40-49, 2018. Disponible (febrero 2021): https://www.researchgate.net/publication/328792735_Una_Propuesta_de_Metodologia_para_la_Migracion_de_Sistemas_Heredados

SOMMERVILLE, I., "Ingenieria del Software," Madrid: Pearson Educación, 2005. (ISBN: 9788478290741).

STANDISH GROUP, "Chaos," Standish Group Report, 2009.

STANDISH GROUP, "Chaos," Standish Group Report, 2015.

TEPPE, W., "The ARNO Project: Challenges and Experiences in a Large-Scale Industrial Software Migration Project," *2009 13th European Conference on Software Maintenance and Reengineering*, Kaiserslautern, pp. 149-158, 2009.

WONG, W.Y., LEE, C.W. and TSHAI, K.Y., "The Importance of a Software Development Methodology in IT Project Management: An Innovative Six Sigma Approach," In: *IEEE Symposium on Business, Engineering and Industrial Applications (ISBEIA2013)*, Kuching, Malaysia, 2013.

WU, B., LAWLESS, D., BISBAL, J., RICHARDSON, R., GRIMSON, J., WADE, V., et al., "The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems", *Proceedings of the 3rd IEEE Conference on Engineering of Complex Computer Systems*, Como, Italy, pp. 200-205, 1997.

Zou, Y., "Incorporating quality requirements in software migration process," *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, Amsterdam, pp. 175-185, 2003.

8. SIGLAS, ABREVIATURAS Y ACRÓNIMOS

8. Siglas, abreviaturas y acrónimos

ARTIST	Advanced software-based seRvice provisioning and migraTIon of legacy Software
BBDD	Base de Datos
CHAOS	Comprehensive Human Appraisal for Originating Software
CMDB	Configuration Management Database
COBOL	COmmon Business-Oriented Language
CRUD	Create Read Update Delete (Crear, Leer, Actualizar y Borrar)
FASMM	Fast and Accessible Software Migration Method
FORTRAN	Formula Translating System
IEEE	Institute of Electrical and Electronics Engineers
IRS	Internal Revenue Service
ITIL	Information Technology Infrastructure Library
JIRA	Herramienta en línea para la administración de tareas de un proyecto, el seguimiento de errores e incidencias y para la gestión operativa de proyectos.
MADIISH	Metodología Ágil de Desarrollo de Software Incremental e Iterativa para la migración de Sistemas Heredados.
MOMOCS	Modeldriven Modernisation of Complex Systems.
PIM	Platform-Independent Model
PL/1	Programming Language 1
PSM	Legacy Platform-Specific Model
RPG	Report Program Generator
SCRUM	Modelo de desarrollo ágil
2PC	Two Phase Commit

9. ANEXOS

9. Anexo: Formularios FASE-1 y FASE-8

9.1. Entrada a los formularios

9.1.1. Descripción

Los formularios están programados en VB de Microsoft. La entrada a ellos se realiza a través de botones definidos en una hoja Excel que, en respuesta al evento “Click”, ejecutan un procedimiento Sub.

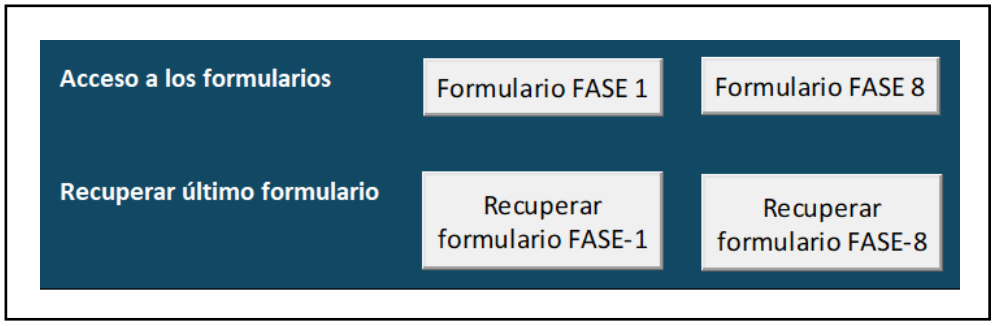


Figura 36 – Botones de entrada a los formularios de FASE-1 y FASE-8.
Fuente: elaboración propia

Botón **Formulario FASE-1**

Abre el formulario de la FASE-1 con los campos inicializados.

Botón **Recuperar formulario FASE-1**

Abre el formulario de la FASE-1 con los últimos valores trasladados a la hoja Excel.

Botón **Formulario FASE-8**

Abre el formulario de la FASE-8 con los campos inicializados.

Botón **Recuperar formulario FASE-8**

Abre el formulario de la FASE-8 con los últimos valores trasladados a la hoja Excel.

9.2. Formulario FASE-1 – Estudio del entorno y la tecnología destino

9.2.1. Descripción del formulario.

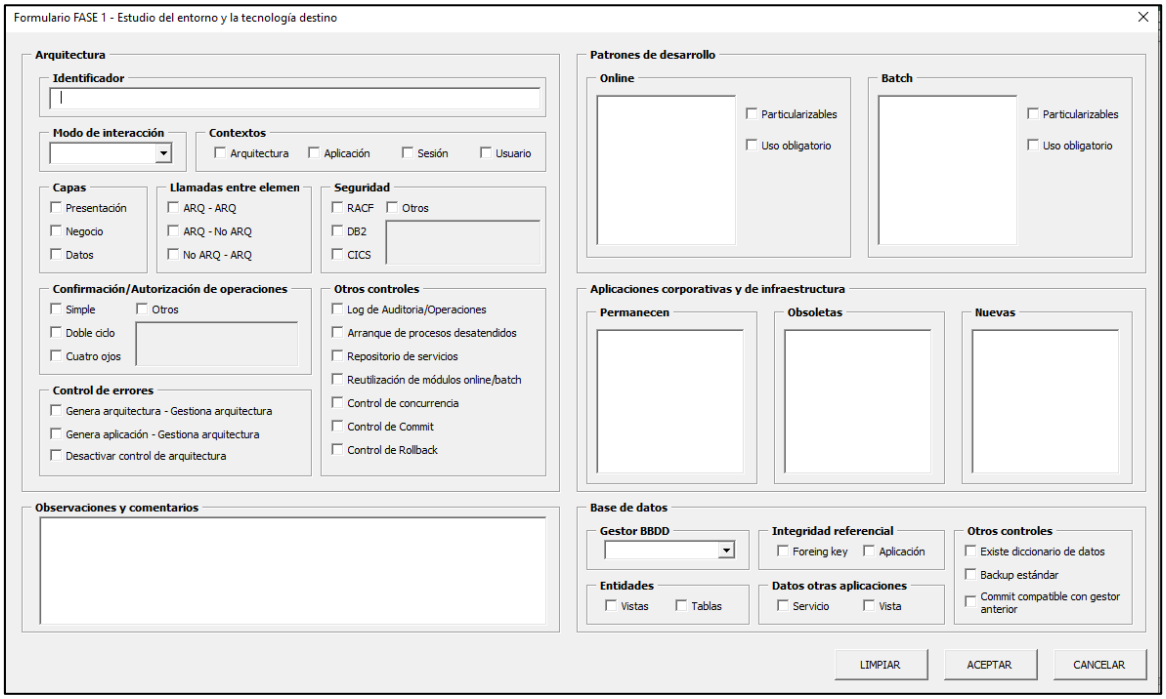


Figura 37 – Formulario de FASE-1.

Fuente: elaboración propia

Arquitectura

- Identificador: nombre y/o breve descripción de la arquitectura.
- Modo de interacción: modo de relación con el usuario
- Contextos: áreas de contexto disponibles en programación
- Capas: capas de las que consta la arquitectura
- Llamadas entre elementos: interacciones permitidas entre programas y rutinas.
    - ARQ-ARQ – Llamante y llamado construidos en arquitectura
    - ARQ-No ARQ – Llamante en arquitectura y llamado externo
    - No ARQ-ARQ – Llamante externo y llamado en arquitectura
- Seguridad: dónde se controla el acceso de usuarios a recursos
- Confirmación/Autorización de operaciones: modos estándar de la arquitectura para solicitar y controlar la confirmación o la autorización a realizar una operación.

    - Simple – Confirmación global de una operación. Se acepta o cancela la operación completa. Ej.: la confirmación de una transferencia acepta el importe, el destinatario y la cuenta.
    - Doble ciclo – Confirmación previa de una parte de la operación y posteriormente se confirma o cancela de manera global. Ej.: en una transferencia se pide primero confirmación del importe. Si se acepta, se solicita la confirmación global.
    - Cuatro ojos – Dos personas distintas deben dar su conformidad a la operación.
    - Otros – Cualquier otro sistema de confirmación que pueda implementar la arquitectura.
- Control de errores: modos estándar de la arquitectura para controlar los errores.
    - Controla arquitectura / Gestiona arquitectura – Errores controlados y generados por la arquitectura. Ej.: registro duplicado, interrumpe la ejecución e informa al usuario sin que el programador deba hacer ningún tipo de implementación.
    - Controla aplicación / Gestiona arquitectura – Errores controlados por la aplicación y gestionados por la arquitectura. Ej.: importe superior al límite máximo, el programador implementa la condición y la arquitectura interrumpe la ejecución e informa al usuario.
    - Desactivar control de arquitectura – Errores controlados y gestionados por la aplicación. Ej.: la condición de registro duplicado no es motivo para generar un error. Se desactiva el control de errores y el programador implementa la acción a tomar.
- Otros controles: otras funciones estándar de la arquitectura.
    - Log de Auditoría/Operaciones
    - Arranque de procesos desatendidos
    - Repositorio de servicios
    - Reutilización de elementos online/batch
    - Control de concurrencia
    - Control de Commit
    - Control de Rollback

Patrones de desarrollo:

- Online: patrones para programación online. Escribir sus nombres en el espacio reservado.
    - Particularizables – Pueden modificarse algunas de las características estándar del patrón. Ej.: modificar la cláusula “order by” en las sentencias de acceso a las tablas que se hayan creado automáticamente.
    - Uso obligatorio – Deben utilizarse siempre en programación online.
- Batch: patrones para programación batch. Escribir sus nombres en el espacio reservado.
    - Particularizables – Pueden modificarse algunas de las características estándar del patrón. Ej.: modificar la cláusula “order by” en las sentencias de acceso a las tablas que se hayan creado automáticamente.
    - Uso obligatorio – Deben utilizarse siempre en programación batch.

Aplicaciones corporativas y de infraestructura

- Permanecen: aplicaciones existentes en el entorno origen y que seguirán estando en el entorno destino. Escribir sus nombres en el espacio reservado.
- Obsoletas: aplicaciones existentes en el entorno origen y que no estarán en el entorno destino. Escribir sus nombres en el espacio reservado.
- Nuevas: aplicaciones inexistentes en el entorno origen y que estarán disponibles en el entorno destino. Escribir sus nombres en el espacio reservado.

Observaciones y comentarios

Área donde escribir un texto libre que aporte información extra sobre el entorno, dé indicaciones especiales, etc.

Bases de datos

- Gestor BBDD: gestor de base de datos que se utiliza en el entorno destino.
- Integridad referencial: modo en el que se implementa la integridad referencial en las aplicaciones del entorno destino.
- Entidades: uso de vistas o tablas en las aplicaciones del entorno destino.

- Datos otras aplicaciones: modo de acceso a los datos almacenados por aplicaciones del entorno destino.
    - Servicio – La aplicación propietaria de los datos publica servicios que son utilizados por las aplicaciones usuarias para acceder a la información.
    - Vistas – Las aplicaciones usuarias definen vistas sobre las tablas de la aplicación propietaria y utilizan estas vistas en sus programas.

Otros controles

- Existe diccionario de datos: en la definición de sus tablas, las aplicaciones deben ceñirse al diccionario de datos de la instalación.
- Backup estándar: se realiza el backup de las bases de datos de manera estándar. Las aplicaciones no necesitan construir sus propios procesos para conseguir copias de respaldo.
- Commit compatible con el gestor anterior: en la interacción de aplicaciones legadas y actualizadas y dentro de una misma transacción, la instrucción commit consolida los datos modificados en el gestor de BBDD legado y el nuevo.

Botón **LIMPIAR**

Inicializa todos los campos del formulario.

Botón **ACEPTAR**

Traslada los valores a una hoja Excel y cierra el formulario.

Botón **CANCELAR**

Cierra el formulario sin trasladar los valores a una hoja Excel.

La hoja Excel a la que se trasladan los valores tiene el diseño del informe que posteriormente se obtiene en formato .pdf. Además de la cabecera (Figura 38), el formulario incluye un apartado para cada una de las áreas definidas: Arquitectura (Figura 39), Patrones de construcción (Figura 40), Aplicaciones corporativas y de infraestructura (Figura 41), Base de datos (Figura 42), y Observaciones y comentarios (Figura 43).

Figura 38 – Informe de FASE-1. Cabecera.

Fuente: elaboración propia

ARQUITECTURA		
Modo de interacción con el usuario		
Contextos disponibles en ejecución		
	Arquitectura	
	Aplicación	
	Sesión	
	Usuario	
Capas de las que consta		
	Presentación	
	Negocio	
	Datos	
Llamadas entre elementos que se pueden implementar		
	Entre elementos integrados en la arquitectura	
	De elementos integrados en la arquitectura a elementos ajenos a ella	
	De elementos ajenos a la arquitectura a elementos integrados en ella	
Productos utilizados en la implementación de la seguridad		
	RACF	
	DB2	
	CICS	
Opciones disponibles para la confirmación y autorización de operaciones		
	Confirmación simple	
	Confirmación de doble ciclo	
	Autorización de 4 ojos (o doble firma)	
	Otros	
Control de errores		
	Errores generados y controlados por la arquitectura	
	Errores generados por la aplicación y controlados por la arquitectura	
	Posibilidad de deshabilitar el control de errores de la arquitectura	
¿Existe un log para el registro de auditoría u operaciones?		
¿Permite el arranque de procesos desatendidos?		
¿Existe un repositorio de servicios?		
¿Permite la reutilización de módulos entre entornos online y batch?		
¿Implementa el control de concurrencia para operaciones de usuario sobre un registro de BBDD?		
¿Tiene el control del commit?		
¿Tiene el control del rollback?		

Figura 39 – Informe de FASE-1. Arquitectura.

Fuente: elaboración propia

PATRONES DE DESARROLLO		
Patrones ONLINE		
	¿Se pueden particularizar?	
	¿Son de uso obligatorio?	
	Relación de patrones online :	
Patrones BATCH		
	¿Se pueden particularizar?	
	¿Son de uso obligatorio?	
	Relación de patrones batch :	

Figura 40 – Informe de FASE-1. Patrones de desarrollo.

Fuente: elaboración propia

APLICACIONES CORPORATIVAS Y DE INFRAESTRUCTURA	
Relación de aplicaciones que seguirán estando disponibles en el entorno destino	
Relación de aplicaciones que dejarán de estar disponibles en el entorno destino	
Relación de nuevas aplicaciones disponibles en el entorno destino	

Figura 41 – Informe de FASE-1. Aplicaciones corporativas y de infraestructura.

Fuente: elaboración propia

BASE DE DATOS		
Entidades con las que se trabaja		
	Tablas	
	Vistas	
¿Cómo implementa la integridad referencial?		
	Mediante Foreing Keys	
	Mediante programación en las aplicaciones	
¿Cómo se accede a los datos de otras aplicaciones?		
	Mediante llamada a servicios	
	Mediante acceso a vistas	
¿Existe un diccionario de datos?		
¿Existe en la instalación un backup estándar de la BBDD?		
El commit de este gestor ¿es compatible con el de la anterior BBDD?		

Figura 42 – Informe de FASE-1. Base de datos.

Fuente: elaboración propia

OBSERVACIONES Y COMENTARIOS

Figura 43 – Informe de FASE-1. Observaciones y comentarios.

Fuente: elaboración propia

Con dos botones de control (Figura 44) se gestiona la inicialización y la generación del informe en formato .pdf.

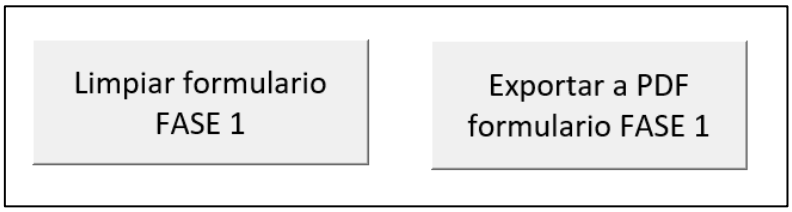


Figura 44 – Informe de FASE-1. Botones de control.

Fuente: elaboración propia

Botón **Limpiar formulario FASE-1**

Inicializa los campos variables del formulario (aquellos en los que se reciben los valores marcados en el formulario).

Botón **Exportar a .pdf formulario FASE-1**

Genera el informe en formato .pdf . El directorio destino se indica en el código de la macro que ejecuta.

9.3. Formulario FASE-8 – Elaboración del plan de implantación y puesta en producción

9.3.1. Descripción del formulario.

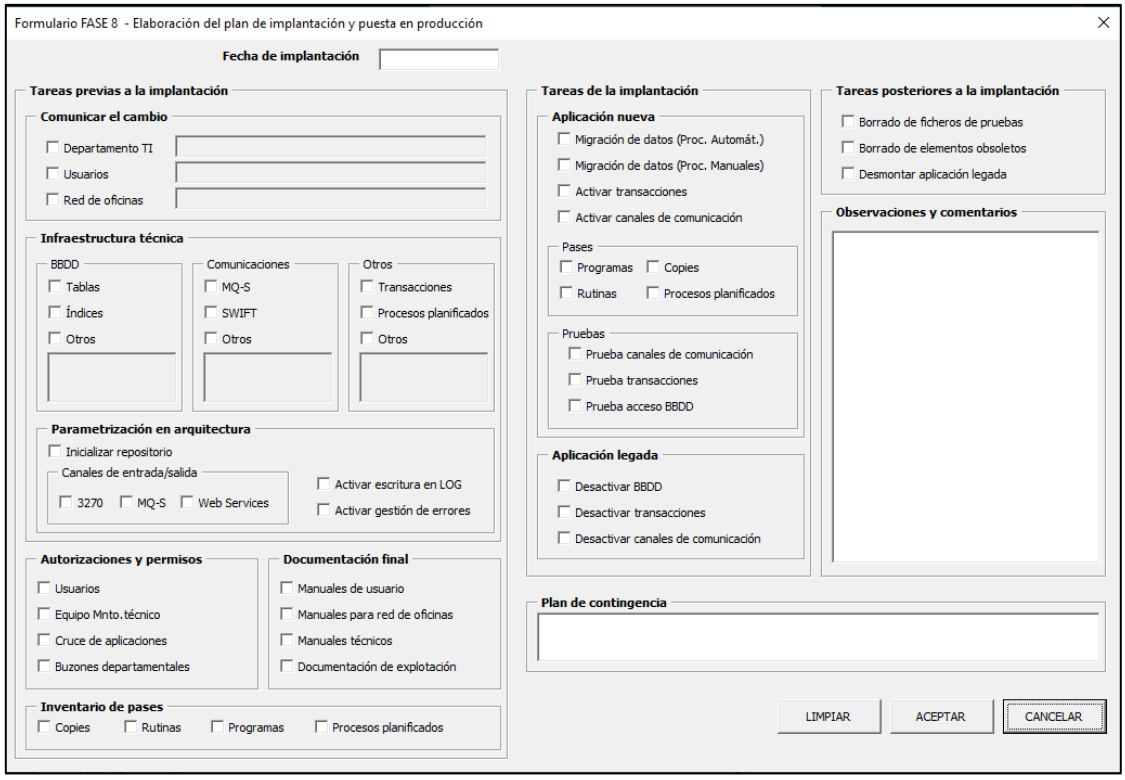


Figura 45 – Formulario de FASE-8.

Fuente: elaboración propia

Fecha de implantación

- Fecha en la que la nueva aplicación construida se instala en el entorno de producción.

Tareas previas a la implantación

Deben realizarse antes de la fecha de implantación.

- Comunicar el cambio:

  Departamentos o personas a las que hay que informar sobre la entrada en producción de la nueva aplicación. Se informan nombres, direcciones de correo electrónico, buzones departamentales, etc.

    - Departamento TI
    - Usuarios
    - Red de oficinas

- Infraestructura técnica:

  Infraestructura que debe definirse en el entorno de producción para el correcto funcionamiento de la nueva aplicación.
    - BBDD. Tablas, Índices, otros (indicar cuáles en el área disponible).
    - Comunicaciones. MQ-S, SWIFT, otros (indicar cuáles en el área disponible).
    - Otros. Transacciones, Procesos planificados, otros (indicar cuáles en el área disponible).

- Parametrización en arquitectura:

  Definición de la nueva aplicación en tablas y ficheros de la arquitectura.
    - Inicializar repositorio – Carga inicial del repositorio de servicios
    - Canales de entrada/salida – 3270, MQ-S, Web Services
    - Activar escritura en Log
    - Activar gestión de errores

- Autorizaciones y permisos:

  Habilitar el acceso a la aplicación en el entorno de producción a quienes vayan a interactuar con ella, ya sean usuarios o aplicaciones.
    - Usuarios
    - Equipo de mantenimiento técnico
    - Cruce de aplicaciones
    - Buzones departamentales

- Documentación final:

  Control de la documentación que debe ser entregada a la finalización del proyecto.
    - Manuales de usuario
    - Manuales para red de oficinas
    - Manuales técnicos
    - Documentación de explotación

- Inventario de pases:

  Elaborar la relación de todos los elementos que deben implantarse en el entorno de producción.

  - Copies
  - Rutinas
  - Programas
  - Procesos planificados

Tareas de la implantación

Deben realizarse el día de la implantación.

- Aplicación nueva:
  - Migración de datos (Proc. Automát.)
  - Migración de datos (Proc. Manuales)
  - Activar transacciones
  - Activar canales de comunicación
  - Pases – Copies, Rutinas, Programas, Procesos planificados
  - Pruebas – Canales de comunicación, Transacciones, Acceso a BBDD
- Aplicación legada:
  - Desactivar BBDD
  - Desactivar transacciones
  - Desactivar canales de comunicación

Tareas posteriores a la implantación

Deben realizarse después de haber completado la implantación de la nueva aplicación en el entorno de producción.

- Borrado de ficheros de pruebas.
- Borrado de elementos obsoletos.
- Desmontar aplicación legada.

Observaciones y comentarios

Área donde escribir un texto libre que aporte información extra sobre el entorno, dé indicaciones especiales, etc.

Plan de contingencia

Acciones que llevar a cabo en el caso de que la implantación de la nueva aplicación en el entorno de producción ocasione problemas en ese entorno. Puede indicarse la ruta a un documento.

Botón **LIMPIAR**

Inicializa todos los campos del formulario.

Botón **ACEPTAR**

Traslada los valores a una hoja Excel y cierra el formulario.

Botón **CANCELAR**

Cierra el formulario sin trasladar los valores a una hoja Excel.

La hoja Excel a la que se trasladan los valores tiene el diseño del informe que posteriormente se obtiene en formato .pdf. Además de la cabecera (Figura 46) y la fecha de implantación (Figura 47), este informe contiene un apartado para cada una de las áreas definidas: Tareas previas a la implantación (Figura 48), Tareas de la implantación (Figura 49), Tareas posteriores a la implantación (Figura 50), Plan de contingencia (Figura 51), y Observaciones y comentarios (Figura 52).

Figura 46 – Informe de FASE-8. Cabecera.

Fuente: elaboración propia

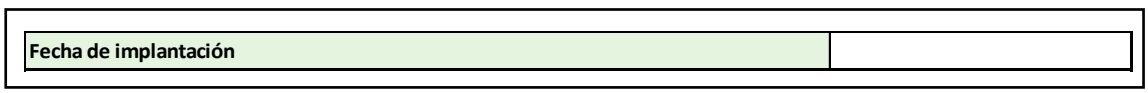


Figura 47 – Informe de FASE-8. Fecha de implantación.

Fuente: elaboración propia

TAREAS PREVIAS A LA IMPLANTACIÓN		
Comunicar el cambio		
	Departamento TI	
	Usuarios	
	Red de oficinas	
Infraestructura técnica		
BBDD	*Tablas*	
	Índices	
Comunicaciones	*MQ-S*	
	SWIFT	
Otros	*Transacciones*	
	Procesos planificados	
Parametrización en arquitectura		
	Inicializar repositorio	
Canales de entrada/salida	*3270*	
	MQ-S	
	Web Services	
	Activar escritura en log	
	Activar gestión de errores	
Autorizaciones y permisos		
	Usuarios	
	Equipo mantenimiento técnico	
	Cruce de aplicaciones	
	Buzones departamentales	
Documentación final		
	Manuales de usuario	
	Manuales para red de oficinas	
	Manuales tecnicos	
	Documentación de explotación	
Inventario de pases		
	Copies	
	Rutinas	
	Programas	
	Procesos planificados	

Figura 48 – Informe de FASE-8. Tareas previas a la implantación.

Fuente: elaboración propia

TAREAS DE LA IMPLANTACIÓN		
Aplicación nueva		
	Migración de datos (Proc. automáticos)	
	Migración de datos (Proc. manuales)	
	Activar transacciones	
	Activar canales de comunicación	
Pases	*Copies*	
	Rutinas	
	Programas	
	Procesos planificados	
Pruebas	*Prueba canales de comunicación*	
	Prueba transacciones	
	Prueba acceso BBDD	
Aplicación legada		
	Desactivar BBDD	
	Desactivar transacciones	
	Desactivar canales de comunicación	

Figura 49 – Informe de FASE-8. Tareas de la implantación.

Fuente: elaboración propia

TAREAS POSTERIORES A LA IMPLANTACIÓN		
Tareas a ejecutar una vez implantada la nueva aplicación		
	Borrado de ficheros de pruebas	
	Borrado de elementos obsoletos	
	Desmontar aplicación legada	

Figura 50 – Informe de FASE-8. Tareas posteriores a la implantación.

Fuente: elaboración propia

PLAN DE CONTINGENCIA
Ruta del documento donde se describe el plan de contingencia

Figura 51 – Informe de FASE-8. Plan de contingencia.

Fuente: elaboración propia

OBSERVACIONES Y COMENTARIOS

Figura 52 – Informe de FASE-8. Observaciones y comentarios.

Fuente: elaboración propia

Con dos botones de control (Figura 52) se gestiona la inicialización y la generación del informe en formato .pdf.

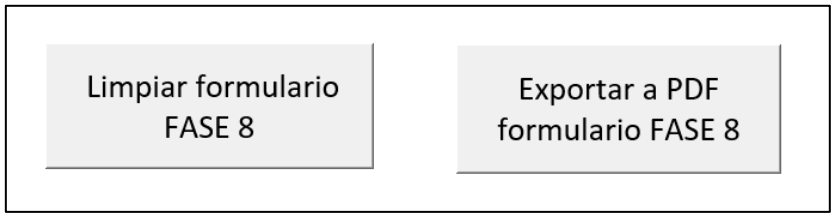


Figura 53 – Informe de FASE-8. Botones de control.
Fuente: elaboración propia

Botón **Limpiar formulario FASE-8**

Inicializa los campos variables del formulario (aquellos en los que se reciben los valores marcados en el formulario).

Botón **Exportar a .pdf formulario FASE-8**

Genera el informe en formato .pdf . El directorio destino se indica en el código de la macro que ejecuta.