30 DE JUNIO DE 2019

Universidad de Oviedo

Metodología de gestión de proyectos outsourcing para desarrollo de software industrial

CONVERTIR UNA EMPRESA DE DESARROLLO DE SOFTWARE PARA LA INDUSTRIA EN UNA EMPRESA ÁGIL Y FIABLE

Autor: Antonio Lago Collar

Director: Miguel A. Vigil

Máster Universitario en Dirección de Proyectos

Resumen

En el momento actual, llamada por los expertos cuarta revolución industrial o industria 4.0 refiriéndose al fenómeno de la digitalización de las cadenas de producción, la inclusión de la realidad virtual, inteligencia artificial o el internet de las cosas, es indispensable para que las empresas industriales puedan seguir siendo competitivas, reduciendo costes de fabricación, aumentando las producciones, aumentando la fiabilidad y mejorando la calidad, la potenciación del departamento de IT.

Dentro de esta automatización de procesos se encuentran las empresas dedicadas al desarrollo de software y servicios que son contratadas por las industrias para la implantación del software que automatizará los procesos de fabricación y ayudarán a aumentar la producción y eficiencia de la industria.

Debido a las características de las empresas que se dedican al outsourcing y las necesidades de la industria, existen ciertas problemáticas a la hora de implantar una u otra metodología de gestión de proyectos ya que la naturaleza de las empresas dedicadas al desarrollo de software es muy diferente a la naturaleza de las empresas industriales.

En este documento se tratará de:

→ Analizar las problemáticas existentes.

→ Se estudiarán diferentes metodologías. Analizando los pros y contras de cada una de ellas, para finalmente justificar el desarrollo de una nueva forma de trabajo, más eficiente con este tipo de proyectos.

→ Se desarrollará una forma de trabajo eficaz, capaz de optimizar los procesos que existen desde que surge la idea o necesidad en la industria, hasta su implantación final por parte de una empresa externa de desarrollo de software.

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

Cada vez son más las empresas del mundo de la industria que deciden externalizar sus servicios informáticos. Para aumentar su productividad y competitividad, las empresas se centran en desarrollar su actividad principal y no desviar recursos a desarrollo y mantenimiento de software. La práctica de outsourcing se generalizó en la crisis económica del 2008 porque era más asequible para las empresas externalizar los servicios que contratar nuevos empleados. (Selectiva, 2018)

La infraestructura IT en una empresa es vital para poder dar unos buenos ratios de productividad y calidad, pero para ello necesitan personal especializado, con experiencia en el sector, lo que resulta caro y complejo. Según el estudio de la consultora americana 451 Research (Research, 2017), al 73,7% de las empresas les resulta complicado reclutar personal para centro de datos o instalaciones.

Si la empresa no puede afrontar este gasto para poder contar con unos servicios IT de calidad deberá externalizar este servicio, esto tiene estos beneficios principales (Gestiopolis, 2018):

- Costes variables, ahorro de dinero.
- Mayor especialización tecnológica de los proveedores.
- Solución a medida y personalizada para la empresa.
- Mayor flexibilidad estructural, es más fácil cambiar de proveedor que la estructura empresarial de integración vertical.
- Responder con rapidez a los cambios del entorno.

Por contra, tiene los siguientes riesgos reconocidos (Montalván, 2009):

- Perdida de know-how por parte de la compañía.
- Riesgo de pérdida de datos cruciales para la empresa.
- Menos agilidad, las soluciones tardan más en ser integradas
- Pérdida del control del servicio
- Riesgo de crear una relación con el proveedor con mucha dependencia

- Riesgo de externalizar actividades vitales cuyo esfuerzo extra para su mantenimiento puede no ser llevado a cabo por el proveedor al no tener la misma involucración con la compañía
- La comunicación y coordinación es más complicada.
- El costo ahorrado puede que no sea el esperado.

En 2004 se publicaba (Ambrojo, 2004), "*India sacude la industria mundial con sus servicios de desarrollo a distancia*", donde se daban estas cifras:

- Un desarrollador en la India gana 6 $/h por los 60 $/h que gana un estadounidense.
- De las 52 compañías mundiales que cumplen el máximo nivel de calidad (SEI-CMM nivel 5), 43 se encuentran en la India.
- India exportó software y servicios por valor de 10.000 millones de dólares y prevé alcanzar los 50.000 millones dentro de cinco años según Nasscom (asociación empresarial).
- Según Goldman Sachs, en los últimos 3 años 200.000 empleos de servicios se han migrado de EEUU a la india

Pero en realidad se superaron las expectativas, como se puede ver en este gráfico que muestra el crecimiento del outsourcing a nivel mundial (Department, 2019)

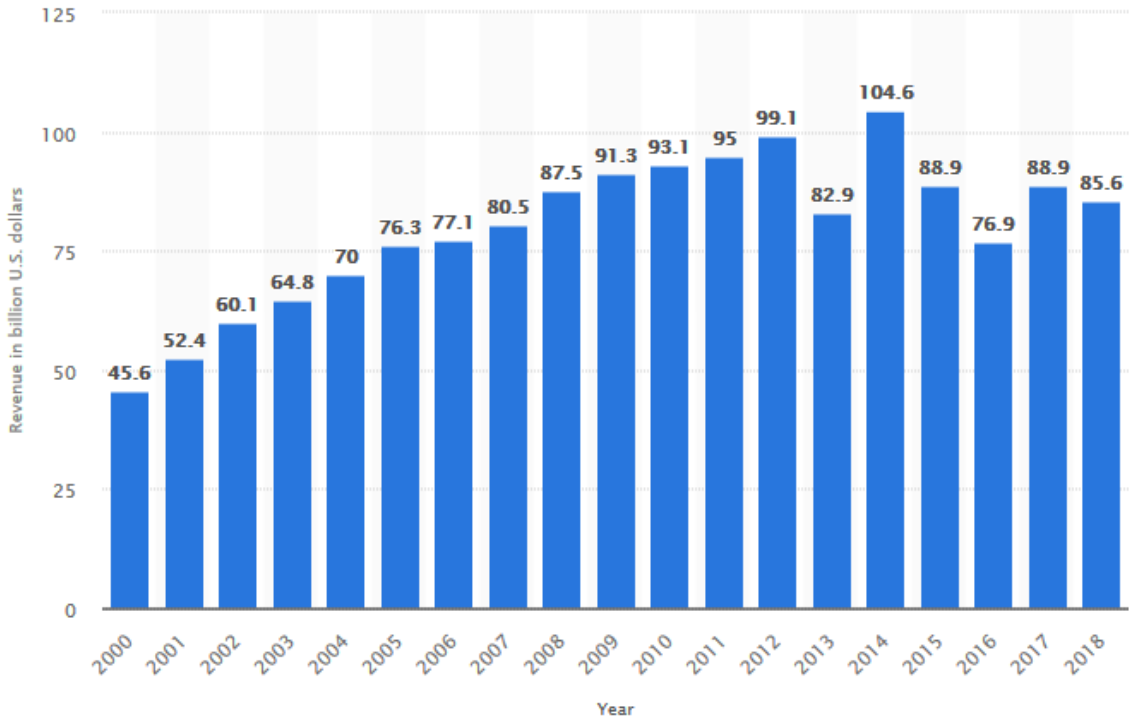


Ilustración 1.1. Mercado global de externalización IT (Department, 2019)

Sin embargo, el outsourcing con empresas extranjeras tiene muchas dificultades, a pesar de que existen empresas de mucha calidad, pero según (Delsol, 2012) estos son los principales problemas de trabajar con empresas de la India que es el principal exportador de outsourcing a nivel mundial:

- Comunicación, no es sencillo mantener una buena comunicación, no como con alguien nativo.
- Falta de innovación, aunque en la India se desarrolla código de gran calidad, no destacan por encontrar nuevas ideas y aportar soluciones nuevas.
- En ese abaratamiento de costes, existe cierto desbordamiento del sector por lo que también se introduce gente que no está capacitada para desarrollar software de calidad.
- Poca experiencia, en general programadores muy jóvenes.
- Por su cultura, son desorganizados y es difícil motivarlos.
- No son gente preocupada por los plazos de entrega.

A escala regional, es habitual encontrarse con empresas de outsourcing que se forman alrededor de las grandes multinacionales industriales, compañías que intentan proveer de servicios software a bajo coste, lo que conlleva al desarrollo de productos software con mano de obra en muchos casos, joven o con poca experiencia en el sector, donde los procedimientos empleados para poder dar productos con la calidad exigida en la industria son vitales.

El outsourcing a escala regional también suele adolecer de problemas de comunicación y alejamiento con el usuario final a la hora de gestionar al cliente y reunir las especificaciones para entender su necesidad. Esta sería la cadena normal de comunicación a la hora de expresar una necesidad:

Los proyectos de desarrollo software para la industria tienen como variable con más peso la **calidad**, ya que una mala calidad del producto puede acarrear en el mejor de los casos unos costes enormes por mala fabricación del producto, y en el peor de los casos problemas de seguridad y de salud graves.

Un ejemplo podría ser un proyecto que consistiese en un proceso de enjuiciamiento automático de carriles. Tiempo más tarde a su implantación, descarrila una máquina, y se detecta que el descarrilamiento fue debido a un defecto en el carril. Los cuantiosos costes para la empresa suministradora del carril no serían comparables, a los costes de que el software dedicado a esa detección de errores se hubiese retrasado dos o tres meses sobre el desarrollo previsto inicialmente y con ello se hubiese evitado el bug que causó el descarrilamiento.

En este tipo de proyectos, no es tan crítico el precio, ya que suelen ser costes muy asumibles por la industria en comparación con los retornos que proporcionan, ni tampoco el tiempo, si se considera que un desarrollo rápido pueda acarrear problemas de calidad. Por lo que, "*hacer las cosas bien y a la primera*", cobra especialmente importancia en el mundo de la industria.

Tampoco se debe pasar por alto, que no siempre las faltas de calidad en el software son motivadas por la variable tiempo, por lo que es importante analizar las distintas variables que afectan a esta falta de calidad final.

En los proyectos outsourcing, se da la casuística de que existe mucha distancia entre el usuario final, y el propio desarrollador. El desarrollador no conoce el trabajo del operador y tiene muy pocos conocimientos sobre el negocio del cliente, y por su parte el operador tiene dificultades para expresar que necesidades reales tiene y que es lo que el desarrollador de software podría hacer por él.

Esta distancia es además incrementada cuando se trata de una empresa de outsourcing perteneciente a otra cultura o sociedad.

Esto hace que la gestión de proyectos sufra dificultades para entender de verdad la necesidad del cliente y poder llevar el proyecto a buen término.

Con objeto de garantizar la consecución de un producto de calidad, con los plazos y costes previstos, nacen las metodologías de gestión de proyectos que han sido muy profusamente adoptadas por parte de la industria del software.

Existen principalmente dos tipos de metodologías en el mundo del software, las tradicionales o en cascada y las ágiles. Dentro de estos tipos, son diversas las metodologías existentes y muchos estudios comparativos entre ellas como por ejemplo (Navarro, 2018), (Martínez, 2013), (Canal, 2016), entre otros.

Las más utilizadas en el mundo por las principales empresas y gobiernos son por un lado Prince 2 como principal metodología en el mundo de las tradicionales, y por otro lado Scrum en las metodologías ágiles.

1.1 **Motivación y objetivos de este TFM**

La motivación para la realización de este estudio se debe a que, por la experiencia del autor, tras años dirigiendo proyectos para el sector industrial, y aplicar diferentes metodologías para la gestión de estos. Se ha percibido una carencia en las metodologías para poder hacer una gestión completa y de calidad, con los recursos del sector. Por ello este estudio que intenta conocer a fondo las diferentes metodologías usadas en el mundo, analizarlas y comprobar cuál sería su aplicación más eficiente para mejorar:

- La calidad del producto final.
- El mantenimiento del mismo y la fiabilidad.
- La agilidad con la que se responde a las necesidades del cliente.
- La gestión de equipos y su motivación.
- La capacidad para llegar a la necesidad real del cliente.
- El aporte de valor a la industria, innovación.
- La comunicación entre los implicados, especialmente el usuario final.

Durante este trabajo se analizan las distintas metodologías empleadas a la hora de desarrollar software para la industria y su aplicabilidad para la gestión de este tipo de proyectos. Finalmente se propondrá una metodología específica, mejor adaptada a los proyectos outsourcing de software industrial y por último se ilustrará su aplicación con un caso real.

2 Principales metodologías para desarrollo de software

En los noventa aparecen las metodologías ágiles, en un intento de mejorar los resultados de los proyectos software en cuanto a tiempo, coste y calidad. Hasta ese momento se usaban metodologías predictivas o waterfall.

Según el informe Chaos (Group S. , 2015) sólo el 29% de los proyectos de software terminan bien a la primera.

	2011	2012	2013	2014	2015
EXITOSOS	29%	27%	31%	28%	29%
MODIFICADOS	49%	56%	50%	55%	52%
FALLIDOS	22%	17%	19%	17%	19%

Tabla 2.1. Resolución de proyectos de FY2011-2015 (Group S. , 2015)

Y tiene especial importancia el tamaño de estos proyectos, cuando más grandes, más dificultades para acabar exitosamente.

	EXITOSOS	MODIFICADOS	FALLIDOS
Muy largos	2%	7%	17%
Largos	6%	17%	24%
Medianos	9%	26%	31%
Moderados	21%	32%	17%
Pequeños	62%	16%	11%

Tabla 2.2. Proyectos en función de su tamaño FY2011-2015 (Group S. , 2015)

Este informe, también hace una distinción entre los proyectos llevados a cabo mediante metodologías predictivas o (waterfall) y a través de metodologías agiles. Saliendo estas últimas mucho mejor paradas en el resultado final de los proyectos.

TAMAÑO	METODOLOGÍA	EXITOSOS	MODIFICAD	FALLIDOS
Cualquier tamaño	Ágil	39%	52%	9%
	Cascada	11%	60%	29%
Largos	Ágil	18%	59%	23%
	Cascada	3%	55%	42%
Medianos	Ágil	27%	62%	11%
	Cascada	7%	68%	25%
Pequeños	Ágil	58%	38%	4%
	Cascada	44%	45%	11%

Tabla 2.3. Resolución de proyectos software FY2011-2015 (Group S. , 2015)

2.1 **Metodologías predictivas**

Algunas de las metodologías predictivas son: CMM-SW, CMMI, PMBOK, PRINCE 2.

En estas metodologías la descripción de lo que se desea obtener está disponible al inicio del proyecto y durante su ejecución se gestiona su cumplimiento. Es una disciplina formal que trata la planificación, ejecución, seguimiento y control a través de procesos sistemáticos y repetibles, para alcanzar las especificaciones definidas en alcance, plazo y coste.

Trata de resolver los planteamientos iniciales cumpliendo con el plazo y coste preestablecido (Menzinsky, 2016).

Se realiza la planificación del desarrollo en tareas, agrupadas en fases. Cada fase tiene asociado recursos con entregables claros, el conjunto de las fases se conoce como ciclo de vida.

Ilustración 2.1. Conceptos de gestión de proyectos (Menzinsky, 2016)

El trabajo se divide en fases, que comienzan al terminar la anterior (waterfall), las más comunes son requisitos, análisis, diseño, codificación, pruebas e implementación.

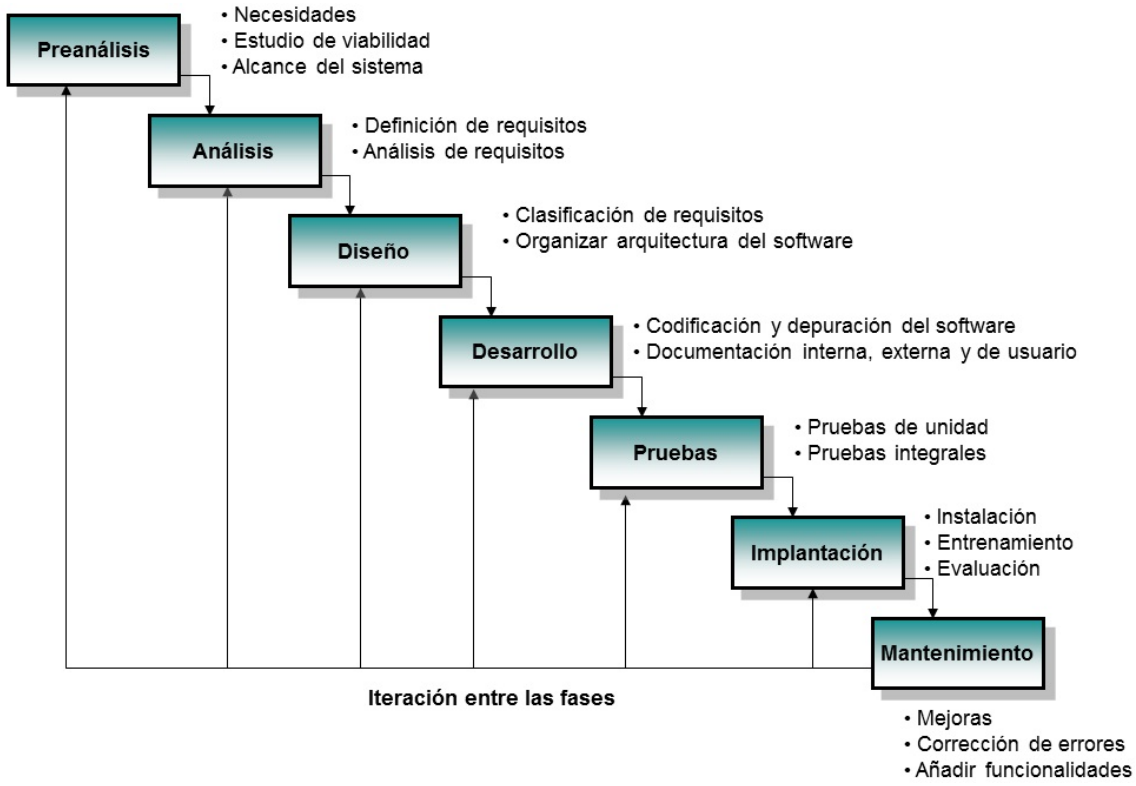


Ilustración 2.2. Modelo cascada (bosco, s.f.)

El conocimiento se basa en los procesos y estos procesos son los responsables de la calidad del resultado, es decir, **la calidad del resultado depende de la calidad de los procesos empleados**.

El objetivo es ofrecer resultados predecibles: desarrollo del producto previsto, en el tiempo previsto e invirtiendo el dinero y los recursos previstos. Los referentes son PMI en su vertiente predictiva (ya existe una certificación PMI para proyectos ágiles) e IPMA y los modelos de procesos CMMI, ISO 15504 y SPICE.

2.1.1 PRINCE2

Dentro de las metodologías predictivas en el mundo del software, una de las más prestigiosas es PRINCE2 (Böhm, 2009), "PRojects IN Controlled Environments" con origen en la oficina de comercio del gobierno del Reino Unido (OGC, 2002). QRP empresa certificadora en PRINCE2 habla de 450.000 directores y gestores certificados en más de 150 países del mundo, con 140 institutos de formación acreditados.

Es usado por empresas como DHL, Vodafone, British American Tobacco, el Gobierno Británico, Naciones Unidas, entre otros.

Según datos del Grupo ILX, organización acreditada de consultoría y formación, este método es el más utilizado del mundo, siendo su tasa de crecimiento del 25% anual de forma constante (Group I. , 2019).

Es una metodología escalable de manera que no es necesario usar todas las partes de la metodología, sino que se adapta al tipo de organización y proyecto.

Al contrario de otras metodologías como PMI, Prince2 es un método para la dirección de proyectos que se basa en productos, desarrollamos un proyecto para conseguir un producto que tenemos claro desde el primer momento (MDAP Executive Master Project Management, 2019).

Sus elementos clave son 7 procesos más 7 principios más 7 temáticas. Divide el proyecto en fases lo que hace que sea más manejable, más amigable y facilita el control eficiente de los recursos.

Ilustración 2.3. Principios Prince2 (Suárez, 2010)

2.1.2 Principios de Prince2

Para que un proyecto sea considerado Prince2 debe cumplir 7 principios.

2.1.2.1 Justificación comercial continua del proyecto

Un proyecto solo comienza si está justificado comercialmente. Se hace a través de un Business Case, y se mantiene durante todo el ciclo de vida del proyecto.

2.1.2.2 Aprender de la experiencia

La experiencia es necesaria para no repetir los mismos errores, por eso se registra en las lesson learned o registro de aprendizaje del proyecto.

2.1.2.3 Roles y responsabilidades

Los proyectos se ejecutan en una organización bien definida, donde está el equipo de proyecto, los proveedores, patrocinadores, actores interesados en el proyecto, clientes, usuarios finales, equipo directivo, etc.

Directivos y ejecutivos.

La empresa es la responsable del proyecto y es la que debe asegurarse de que el proyecto es económicamente viable. Fija los límites de tolerancia, aprueba el comienzo y fin del proyecto y monitoriza las finanzas.

Junta de proyecto.

La designa los directivos de la empresa para dirigir la gestión de los proyectos, es responsable del éxito o fracaso del proyecto. Sus responsabilidades específicas es la autorización de las planificaciones principales, comienzo y fin de cada etapa, autorizar desviaciones con respecto a lo especificado en el plan de etapa.

Usuario senior.

Representa a los usuarios finales del producto y es el enlace con el equipo de trabajo. Son los responsables de la especificación de las necesidades y monitorizan que la solución este acorde a estas necesidades en términos de calidad funcional y facilidad de uso.

Proveedor senior.

Representa los intereses del equipo de desarrollo del proyecto. Debe tener suficiente autoridad como para adquirir aquellos recursos necesarios para el proyecto. Forma parte de la junta del proyecto.

Se encargan de monitorizar los cambios en el proyecto, los riesgos potenciales, vigilar los procedimientos de control de calidad y aconsejar sobre estrategias técnicas o métodos de diseño.

Jefe de proyecto.

Dirige el proyecto en su día a día, representa a la junta de proyecto. Su responsabilidad principal es asegurar que el proyecto genera los productos requeridos, con los estándares de calidad establecidos y las restricciones de tiempo y coste. También es el responsable de que los beneficios en el caso de negocio se consigan.

Jefe de equipo.

Puede ser que el jefe de proyecto se encargue de estas funciones o puede derivar en una persona diferente. Su responsabilidad principal es la de asegurar la producción de los

productos definidos por el jefe de proyecto, cumpliendo con la calidad, el tiempo y el coste.

Responsable de monitorizar el rendimiento y los productos del proyecto.

Se requiere que sea un papel independiente del jefe de proyecto, no puede ser delegado por él, es una responsabilidad que suele recaer sobre el ejecutivo, usuarios senior y proveedores senior.

Es el encargado de garantizar que los riesgos están siendo controlados, asegurar la fidelidad al caso de negocio, el enlace entre el proveedor y el consumidor.

2.1.2.4 Gestión del proyecto por fases

Se divide en pre-proyecto – inicio – entrega y final – post proyecto.

Las entregas se realizan al final de cada fase.

2.1.2.5 Gestión por Excepción

Indica la tipología de los que intervienen en el proyecto, en tres niveles:

- Nivel gobierno, la dirección que soporta la orientación del proyecto.
- Nivel gestión, director del proyecto y su equipo.
- Nivel entrega, el ambiente técnico.

Cada nivel cede al nivel inmediatamente inferior responsabilidades. Los niveles inferiores rinden cuentas a los niveles superiores.

"yo hago lo que no haces tú, pero de lo que tú haces, en realidad, yo soy el responsable último" (MDAP Executive Master Project Management, 2019).

2.1.2.6 Enfoque hacia la entrega de productos

En el proyecto siempre está presente el producto final.

2.1.2.7 Adaptación al cambio

Indica que se hay que adaptar a cada una de las circunstancias.

2.1.3 Temáticas de Prince2

En Prince2 hay siete temáticas frente a las 15 de PMBOK (teniendo en cuenta las 5 de grupos de procesos). PMBOK duplica las áreas de conocimiento. En Prince2 se tratan aspectos más directivos mientras que en PMBOK son más detallados.

Define las temáticas como los ámbitos o aspectos del proyecto que necesitan ser gestionados durante toda la vida del proyecto. Cada una proporciona una metodología para esa área específica.

Ilustración 2.4. Temáticas Prince2 (Suárez, 2010)

Cada temática describe cuatro puntos:

- Por qué
- Definición
- Enfoque
- Intervención

2.1.3.1 Bussines Case

Justificar el proyecto e indicar el por qué el proyecto debe continuar. Se supervisa continuamente para seguir justificándolo.

Este documento forma parte de la inicialización del proyecto aunque se revisa durante toda la vida del proyecto.

Se contemplan estos puntos:

- Razones para la realización del proyecto
- Beneficios, deben poder medirse.
- Opciones consideradas, consideraciones tenidas en cuenta antes de llegar a esta solución.
- Coste y tiempo
- Evaluación de la inversión, relación coste-beneficio a la largo del proyecto.
- Análisis GAP y de sensibilidad, para determinar la robustez del caso de negocio, indicando las incertidumbres y riesgos del proyecto.
- Riesgos principales

2.1.3.2 Organización

Estudiar el conjunto de agentes del proyecto y su rol dentro del mismo.

Lo habitual en Prince2 son estas cuatro capas.

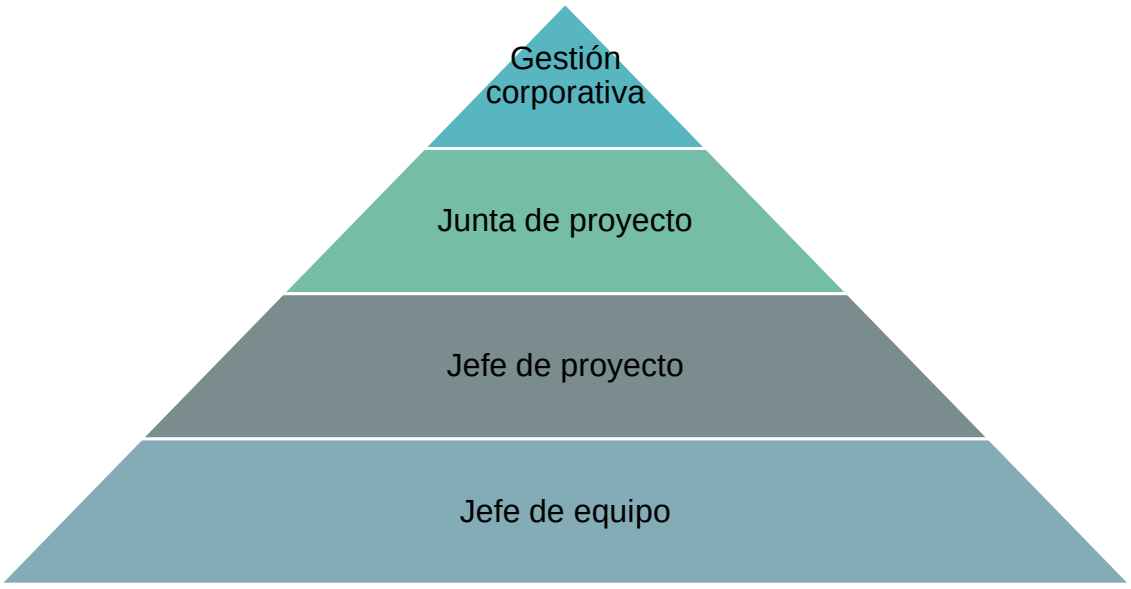


Ilustración 2.5. Capas de organización del proyecto (Suárez, 2010)

2.1.3.3 Calidad

Se debe verificar que lo producido se ajusta al proyecto. Tras el acuerdo con el cliente de los criterios de calidad, se establecen los estándares que se utilizarán y el personal responsable de que estos se cumplan. Estos requisitos de calidad deben añadirse a los planes para integrarlos en el proyecto.

Las expectativas del cliente deben ser medibles, después se decide la forma en la que el proyecto alcanzará estas expectativas.

Cada etapa tiene su plan de calidad con un responsable que verificará esta calidad individual. Esta descripción constará de:

- Título
- Propósito
- Componentes del producto
- Formato
- Criterio de calidad
- Método de calidad, incluyendo las pruebas que se llevarán a cabo para verificar que se cumplen los criterios de calidad.

Se tiene un registro de calidad con el resumen de las pruebas planificadas, las llevadas a cabo y los resultados.

2.1.3.4 Cambio

Cambios que implican reorientar el proyecto. Sería la gestión del cambio en PMBOK.

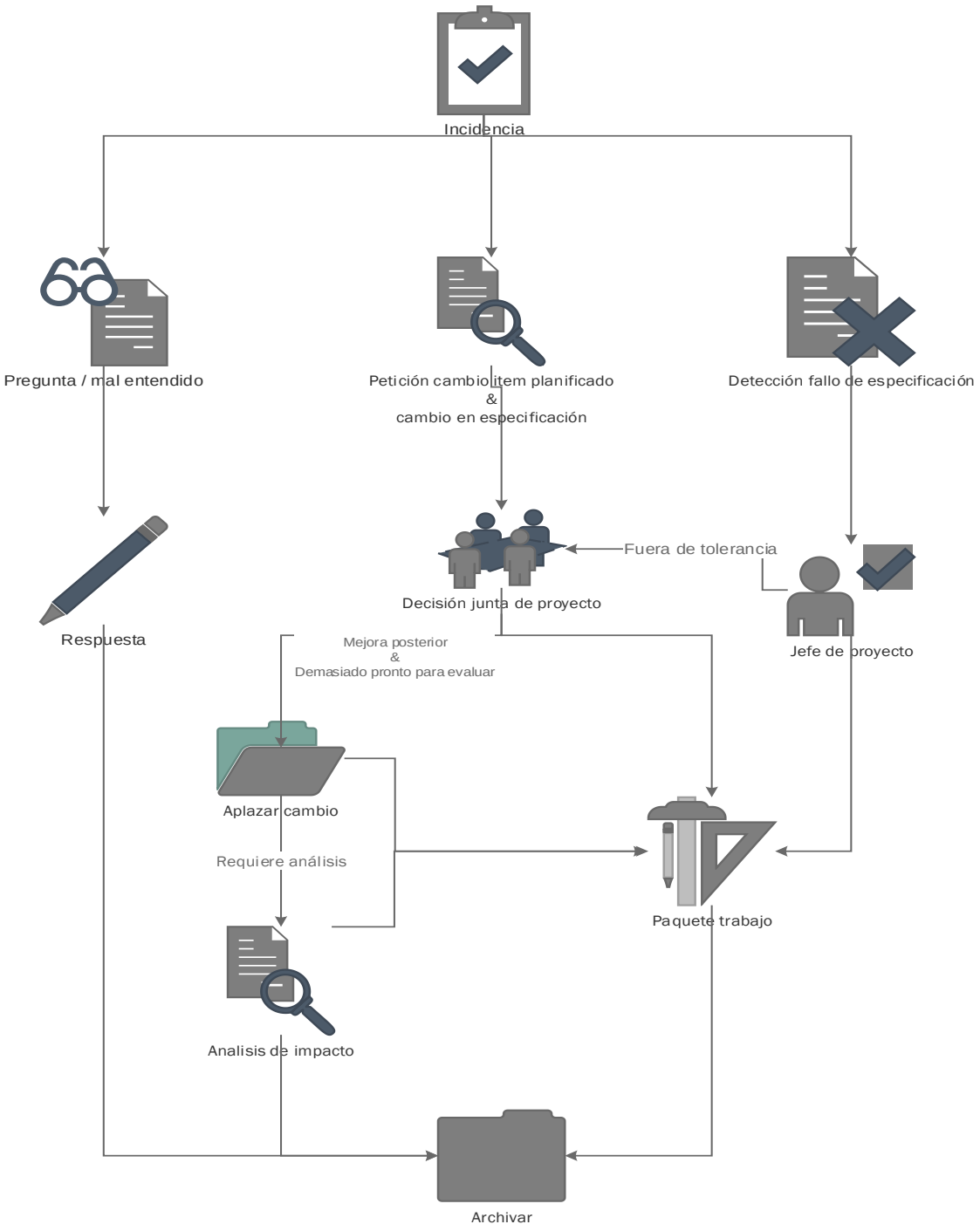


Ilustración 2.6. Gestión cambios (Suárez, 2010)

Petición de cambio

Proponen modificaciones por lo que requieren un análisis previo denominado análisis de impacto que llevará a cabo el equipo de trabajo con conocimientos adecuados.

Para que se llegue a implementar debe haberlo aprobado el jefe de proyecto o la junta de proyecto en función de la tolerancia.

Fallo en la especificación

Se procede al igual que en el caso anterior, se envía el documento al archivo de configuración, se le asigna un identificador, se envía copia a quien elevó la incidencia y se procede con el análisis de impacto.

Las decisiones deben quedar documentadas en el registro de incidencias, se debe enviar una copia a quien la originó y se debe actualizar el registro de configuración y de incidencias.

2.1.3.5 Riesgos

Identificación de los riesgos, planes de contingencia, responsabilidades, impacto, etc.

Un riesgo está definido como una incertidumbre sobre el resultado, tanto una oportunidad positiva como una amenaza negativa. Se deben medir esos riesgos, identificarlos y comprobar cuál es la probabilidad de que realmente ocurran y que acciones emprender en ese caso.

El objetivo es evaluar la exposición del proyecto a los riesgos y planificar que acciones se pueden tomar para que ese riesgo esté en niveles aceptables, con un coste aceptable.

Se debe también definir la tolerancia a riesgos, en base a estos parámetros:

- Completar el proyecto en el margen de tiempo adecuado
- Completarlo dentro del coste establecido
- Conseguir la calidad del producto establecida
- Conseguir el alcance definido durante el Bussines case.

La gestión del riesgo es responsabilidad de la junta del proyecto y el jefe de proyecto. La junta tiene estos compromisos:

Notificar al jefe de proyecto cualquier exposición a riesgos externos que tenga el proyecto.

Tomar decisiones sobre acciones propuestas por el jefe de proyecto.

Notificar a la gestión corporativa sobre cualquier riesgo que pueda afectar a que el proyecto respete las restricciones impuestas por dicho órgano.

Ilustración 2.7. Ciclo gestión de riesgos (Autor)

2.1.3.6 Progreso

Se analiza y evalúa continuamente el proyecto. Cómo estamos, dónde vamos, ¿deberíamos continuar?

El control del proyecto depende del tipo de proyecto y se divide en etapas de esta forma resulta más fácil detallar el alcance del proyecto de una etapa que de todo el proyecto.

La junta decide si sigue con el proyecto basándose en:

- Los resultados de la etapa actual
- Una evaluación del plan de la etapa siguiente

- Una verificación sobre el impacto de la etapa siguiente en el proyecto general.
- Una confirmación de que los riesgos son asumibles.

Cada etapa debe ser aprobada por la junta de proyecto a su finalización.

Existe una desviación del plan, en la que no es necesario consultar al nivel superior de autoridad, se trata de la tolerancia. Esta tolerancia la establece la directiva al inicio del proyecto con el jefe de proyecto y este a su vez con el jefe de equipo.

De esta forma se evita elevar continuas incidencias a la junta de proyecto.

El jefe de proyecto debe mantener un registro diario de los eventos significativos que complementa el plan de proyecto.

Al comienzo de cada semana el jefe de proyecto supervisa el plan de etapa, el registro de riesgos, el de incidencias y el archivo de calidad para decidir acciones a lo largo de esa semana.

2.1.3.7 Planes

Se refiere al alcance y el cronograma. Describe cómo, cuándo y quién es responsable de conseguir una serie de objetivos.

Debe contemplar:

- Los productos a ser desarrollados para conseguir una determinada meta
- Los pasos requeridos para conseguir dichos productos
- La secuencia de dichos pasos
- Interdependencia entre los productos
- Estimación de tiempo
- Quién lleva a cabo cada etapa y cuándo
- Dónde se debe aplicar el control
- Muestra con antelación si un objetivo puede ser conseguido
- Qué recursos se necesitan para llevar a cabo el trabajo
- Cuánto tiempo llevará realizar el trabajo

- Proporciona una base para evaluar los riesgos involucrados en el trabajo

Para esto se debe hacer un desglose del producto (PBS) y un diagrama de flujo del producto (PFD)

Plan de proyecto.

Se crea al comienzo del proyecto y forma parte de la inicialización. Es obligatorio. Sirve a la junta del proyecto para ver una vista general a alto nivel de lo que será el proyecto. Permite saber:

- Cuánto durará el proyecto
- Cuáles serán los entregables más importantes
- Una aproximación de cuándo se harán las entregas
- Qué recursos y trabajadores se necesitarán para conseguir los objetivos fijados en el plan.
- Cómo se ejercerá el control
- Cómo se mantendrá la calidad
- Qué riesgos existen en la solución tomada para el proyecto.

Plan de etapa.

Se lleva a cabo al finalizar la etapa anterior y se responsabiliza el jefe de proyecto para gestionar el día a día del proyecto. Si el proyecto es pequeño no es obligatorio este plan.

Contiene:

- Descripción del plan
- Resumen del plan y antecedentes
- La aproximación que se pretende hacer a la implementación
- Cualquier otro detalle como diagrama de Gantt, dependencias del proyecto, riesgos.

Plan de equipo

Son opcionales. Se usan a bajo nivel por los jefes de equipo, con una estructura desglosada por actividades específicas.

Plan de excepción

Se produce cuando se prevé que el proyecto sobrepasa los límites de tiempo o coste acordados entre el encargado de planificar y el responsable superior.

Si se aprueba este plan de excepción, este sustituirá al plan actual y tendrá el mismo formato que el anterior.

2.1.4 Procesos Prince2

Cada actividad se describe en un gráfico con procesos, actividades, desencadenantes y productos.

Son ocho procesos distintos, con subprocesos. Todos los proyectos Prince2 tienen que ajustarse a estos ocho procesos.

2.1.4.1 Proceso de puesta en marcha (DP)

Consiste en que la junta del proyecto apruebe el inicio de aquellos proyectos que son viables. Asegura la justificación comercial del proyecto. Nombran al ejecutivo, registra lecciones anteriores, nombra al equipo de gestión del proyecto, selecciona el enfoque del proyecto y elabora el expediente del proyecto.

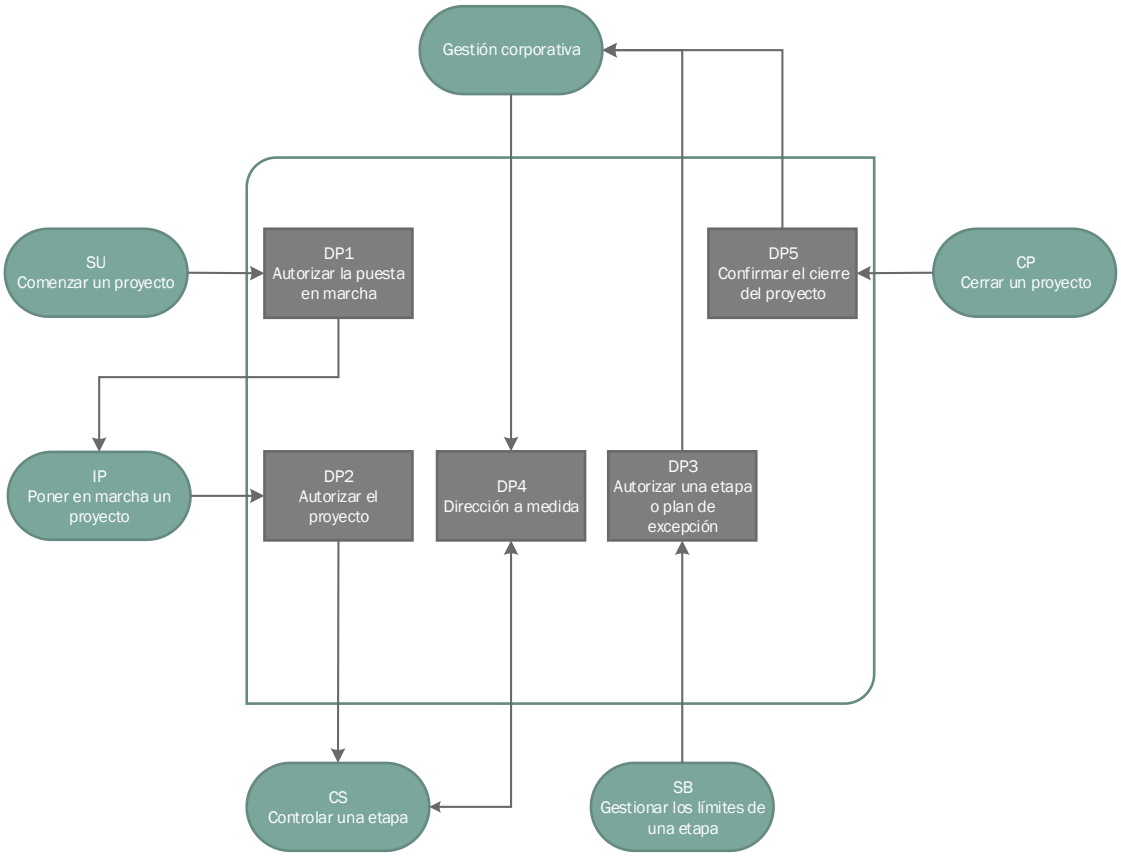


Ilustración 2.8. Subprocesos DP (Suárez, 2010)

Sigue cinco pasos:

- Autorizar la preparación de un plan de proyecto
- Autorizar la puesta en marcha
- Comprobar que el proyecto sigue siendo justificable
- Monitorizar el progreso y aconsejar cuando sea necesario
- Asegurarse de que el proyecto llega a un final controlado

2.1.4.2 Proceso de inicio (SU)

Establece las bases asegurando que la organización entiende el esfuerzo a realizar antes de comprometerse a hacerlo. Su objetivo es:

- Diseñar y designar el equipo de gestión
- Asegurar de que los objetivos están claros

- Decidir el planteamiento que servirá de base para proporcionar soluciones durante el proyecto
- Definir las expectativas de calidad del cliente
- Planificar el trabajo necesario para realizar el contrato tipo entre cliente y proveedor.

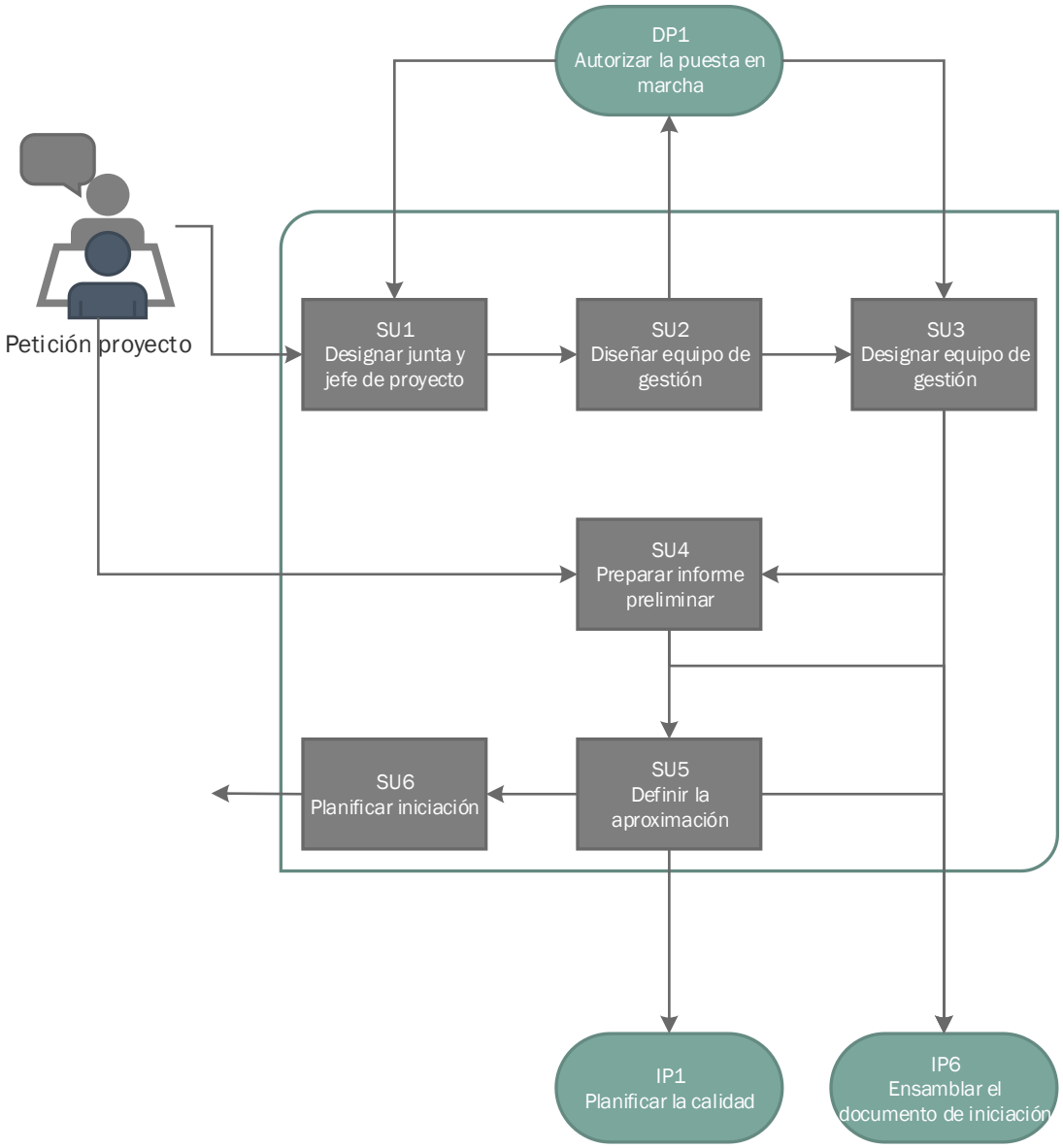


Ilustración 2.9. Subprocesos SU (Suárez, 2010)

2.1.4.3 Proceso de dirección del proyecto (IP)

Facilita que la junta de proyecto tome las decisiones clave y ejerza el control general del proyecto. Se busca que haya una autoridad para iniciar el proyecto, entregar los productos y cerrar el proyecto.

La salida de este proceso es el documento de iniciación del proyecto, que establece la línea base con la cual se podrá medir el progreso y el éxito del proyecto.

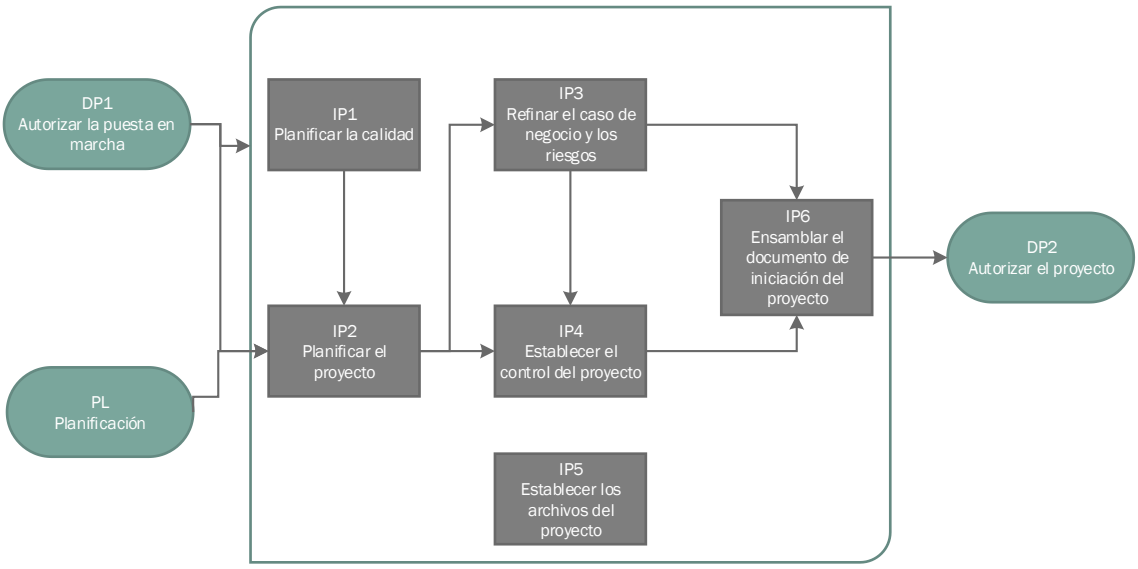


Ilustración 2.10. Esquema del proceso IP (Suárez, 2010)

2.1.4.4 Proceso de control de fase del proyecto (CS)

Entregar los productos, revisar el modelo de negocio y controlar los riesgos. Las actividades son:

- Autorizar el trabajo para que se haga
- Reunir información sobre el progreso de ese trabajo
- Vigilar los cambios
- Revisar la situación
- Generar informes
- Añadir lecciones útiles al registro de lecciones aprendidas
- Tomar cualquier acción necesaria

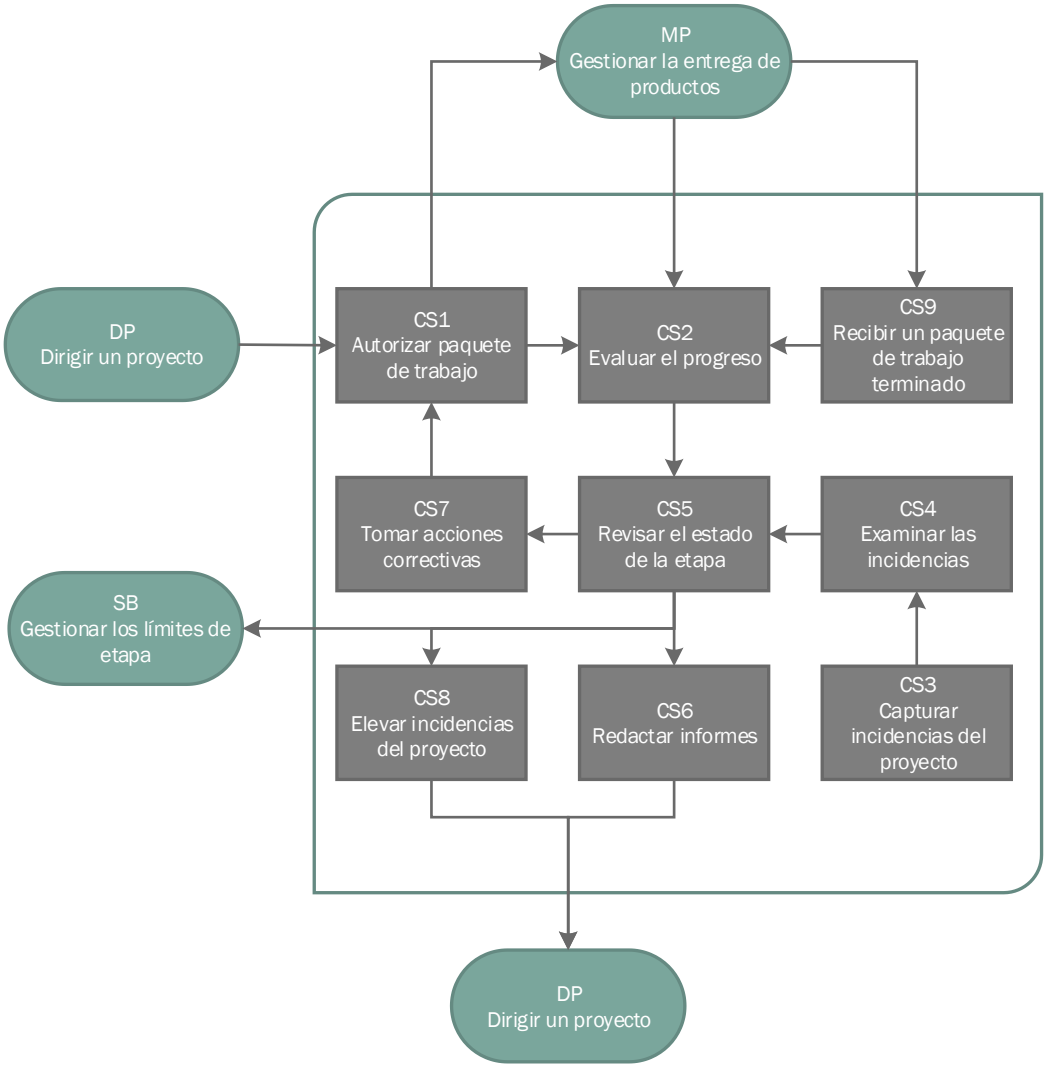


Ilustración 2.11. Subprocesos CS (Suárez, 2010)

2.1.4.5 Proceso de gestión de entrega de productos (MP)

Establece los requisitos formales para acordar, completar y entregar todos los trabajos del proyecto.

El trabajo acordado por el jefe de proyecto y el equipo de gestión se llama paquete de trabajo, incluye fecha de entrega, requisitos de calidad y requisitos de informes. Debe cubrir:

- Asegurar que el trabajo asignado al equipo esté autorizado y acordado
- Planificar el trabajo del equipo

- Asegurar de que el trabajo está hecho
- Asegurar que se cumple con los requisitos de calidad
- Informar del progreso y calidad al jefe de proyecto
- Obtener aceptación de los productos finalizados

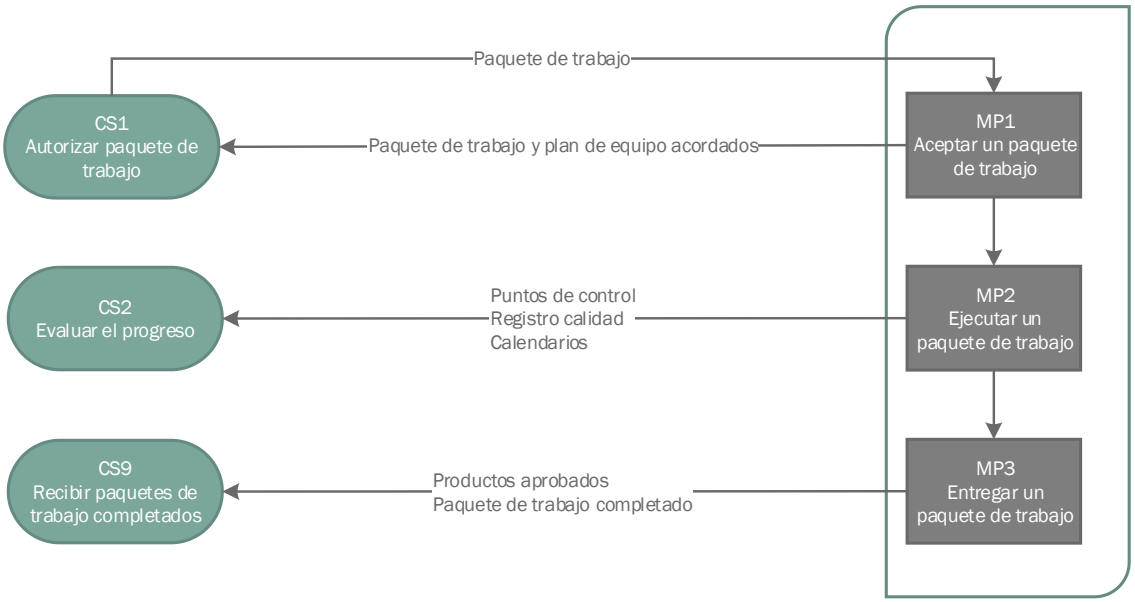


Ilustración 2.12. Subprocesos MP (Suárez, 2010)

2.1.4.6 Proceso de gestión de los límites de fase (SB)

Posibilita que cuando el final de la fase en curso sea inminente, la junta de proyecto revise el desarrollo de la fase en curso y apruebe el plan de la fase siguiente.

- Planifica la siguiente etapa
- Actualiza el plan del proyecto
- Actualiza el caso de negocio
- Actualiza la evaluación de riesgos
- Informa del rendimiento y productos terminados de la etapa que llega a su fin.
- Obtiene la aprobación de la junta del proyecto para continuar a la siguiente etapa

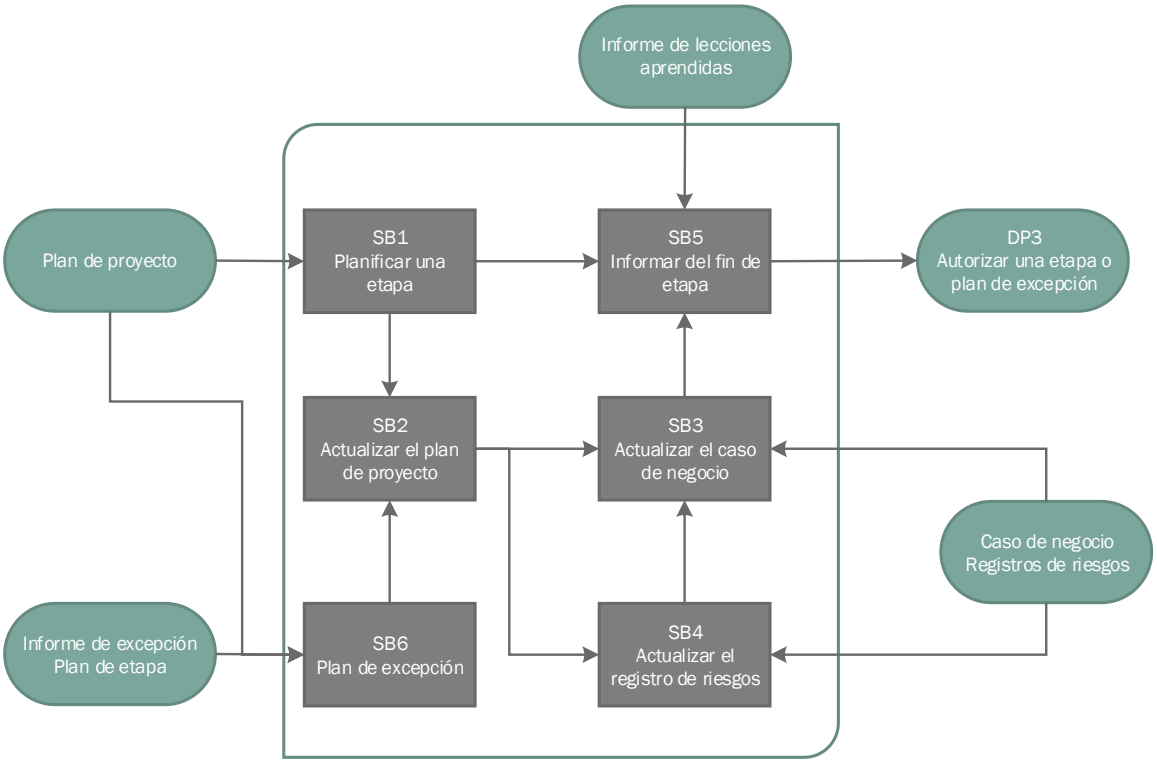


Ilustración 2.13. Subprocesos SB (Suárez, 2010)

2.1.4.7 Proceso de cierre del proyecto (CP)

Proporciona un punto final de control en el que se confirme la aceptación del producto del proyecto y se reconozca que se han alcanzado los objetivos establecidos en la documentación de inicio del proyecto.

Tiene estos objetivos:

- Anotar hasta qué punto se han cumplido los objetivos impuestos al comienzo del proyecto
- Confirmar la satisfacción del cliente con los productos
- Confirmar que el mantenimiento y soporte del producto estén acordados
- Hacer cualquier recomendación para el futuro.
- Registrar las lecciones aprendidas
- Dejar constancia del éxito o fracaso del proyecto
- Preparar un plan que verifique si el producto obtuvo los beneficios esperados.

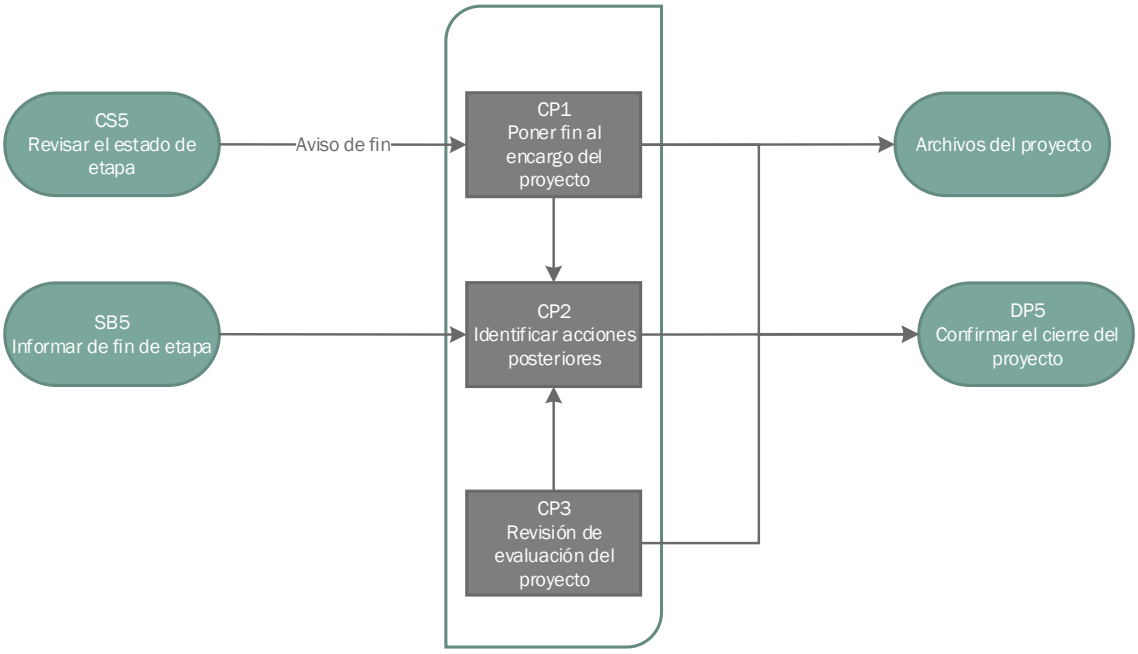


Ilustración 2.14. Subprocesos CP (Suárez, 2010)

2.1.4.8 Planificar (PL)

Es un proceso repetible, usado por los otros procesos cuando se necesita un plan.

- Diseño del plan
- Definición y análisis de los productos
- Identificación de las actividades necesarias y las dependencias
- Estimación del esfuerzo
- Planificación de recursos
- Análisis de riesgos
- Descripción del plan

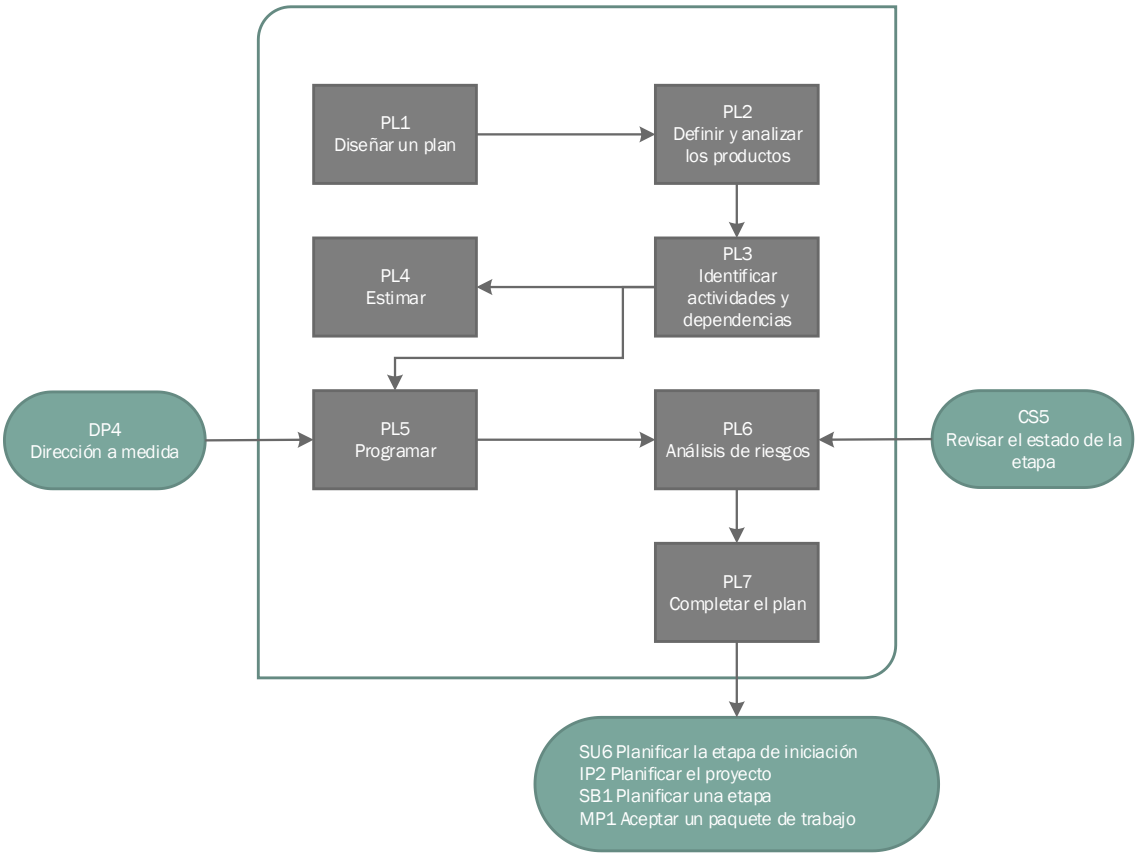


Ilustración 2.15. Subprocesos PL (Suárez, 2010)

2.2 **Metodologías agiles**

Según (Alexander Menzinsky, 2016), ahora se construye el producto al mismo tiempo que se modifican e introducen nuevos requisitos. El cliente parte de una visión, pero el nivel de innovación que se requiere, y la velocidad a la que se mueve el entorno de su negocio, no le permiten prever con detalle cómo será el resultado final. *Quizá ya no hay "productos finales", sino productos en continua evolución y mejora*.

La gestión de proyectos ágil no se formula sobre la necesidad de anticipación, sino sobre la de adaptación continua.

En marzo de 2001, 17 desarrolladores de software fueron convocados por Kent Beck autor del libro "Extreme Programming" (Beck, 2000). Los integrantes de la reunión crearon el "**Manifiesto Ágil**" que intenta huir de los métodos demasiado rígidos y de planificaciones detalladas, en los que se sustentan los métodos tradicionales.

2.2.1 Manifiesto Ágil

- Individuos y su interacción por encima de los procesos y las herramientas.
- Software que funciona por encima de la documentación exhaustiva.
- Colaboración con el cliente, por encima de la negociación contractual.
- Respuesta al cambio por encima del seguimiento de un plan.

2.2.2 Principios del manifiesto ágil

- Satisfacer al cliente con entregas tempranas y continuas de software de valor.
- Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente
- Entrega frecuente de software funcional
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana
- Construcción de proyectos en torno a individuos motivados
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida de progreso
- Los procesos ágiles promueven el desarrollo sostenido.
- La atención continua a la excelencia técnica enaltece la agilidad
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace

- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

2.2.3 Scrum

Dentro de las metodologías ágiles, la más usada en el mundo, es Scrum que se analiza a continuación como referente de este tipo de metodologías.

Este modelo fue definido por Ikujiro Nonaka e Hirotaka Takeuchi en los 80, basándose en el desarrollo de productos de estas fábricas industriales: Fuji-Xerox, Canon, Honda, Nec, Epson, Brother, 3M y Hewlett-Packard (Nonaka, 1986). En su estudio se fijaron en el avance en formación de Scrum de los jugadores de Rugby y de ahí su denominación de Scrum.

Ilustración 2.16. Formación de rugby scrum (Alexander Menzinsky, 2016)

Esta forma de trabajo surgió en empresas con proyectos con requisitos inestables para los que se requería rapidez y flexibilidad.

En 1995 Ken Schwaber presentó “Scrum Development Process”, un marco para el desarrollo de software basado en los principios de Scrum.

Scrum Manager como ambiente de trabajo compuesto por equipos autoorganizados que trabajan de forma ágil. Con autonomía y solapamiento en las fases de desarrollo, compartiendo el conocimiento.

Metodología Scrum

- Previo al juego
  - Planificación
  - Arquitectura de diseño de nivel alto
- Durante el juego
  - Sprints (análisis, diseño, desarrollo)
  - Empaquetar
  - Revisar
  - Ajustar
- Después del juego
  - Cierre

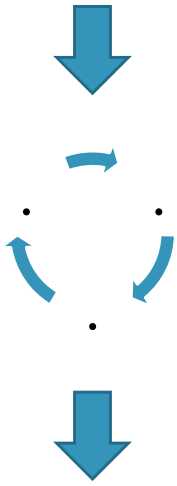


Ilustración 2.17. Scrum Development Process (Alexander Menzinsky, 2016)

Debemos distinguir entre un Scrum técnico, con reglas para desarrollar siguiendo esta metodología y el Scrum asociado a unos valores, que es el concepto original.

Scrum Técnico	Valores Scrum
 Ilustración 2.18. Ken Schwaber y Jeff Sutherland 1995	 Ilustración 2.19. Hirotaka Takeuchi e Ikujiro Nonaka 1986
Reglas definidas	**Valores ágiles**
Roles • Dueño de producto • Equipo de desarrollo • Scrum master	**Valores** • Personas > procesos • Resultados > documentación • Colaboración > negociación • Cambio > planificación

Eventos • El sprint • Reunión de planificación • Scrum diario • Revisión de sprint • Retrospectiva de sprint	**Aplicación** • Incertidumbre • Autoorganización • Fases de desarrollo solapadas • Multiaprendizaje • Control sutil • Difusión del conocimiento
Artefactos • Pila de producto • Pila de Sprint • Incremento	

(Alexander Menzinsky, 2016)

En 2005 Mike Cohn, Esther Derby y Ken Schwaber constituyeron "Scrum Alliance" que difunde un marco de trabajo específico para el desarrollo de software.

Ilustración 2.20. Marco Scrum (Alexander Menzinsky, 2016)

2.2.3.1 Roles

- Propietario del producto

  Suele tratarse del cliente, en el caso de que el cliente no quisiese participar, un miembro del equipo, que conozca bien el producto, normalmente el jefe de equipo asume este rol.

  Es la persona encargada de presentar las historias de usuario de la pila de producto, y priorizarlas.

  Debe dar el nivel de detalle suficiente para que el equipo pueda ser capaz de construir el incremento.

- Equipo de desarrollo

  Se trata de los desarrolladores, diseñadores, analistas… las personas encargadas de crear el producto.

  Pueden proponer sugerencias, modificaciones o soluciones alternativas, durante las reuniones de sprint con el propietario del producto

- Scrum Máster

  Es la persona encargada de:

  - Promover la reunión de Sprint, fijar la fecha y el lugar y asegurarse de que asistan los involucrados.
  - Asegurar que en la reunión estará preparada la pila del producto para debatir sobre ella
  - Moderar la reunión, asegurándose de que se llegan a acuerdos concretos entre el equipo y el propietario del producto.
  - Ayudar al equipo a comprender las necesidades del negocio del cliente.
  - Asegurarse de que la descomposición y estimación del trabajo por parte del equipo es realista, y se han considerado todas las posibles tareas necesarias, como el análisis, las pruebas, la investigación, etc.
  - Al final de la reunión, debe asegurarse de:
    - Que elementos de la pila del producto se van a ejecutar

- El objetivo del sprint
- La pila del sprint con las tareas estimadas
- Duración del sprint y fecha de la reunión de revisión

2.2.3.2 Artefactos

- Pila del producto

No importa si es gestión tradicional o ágil, la pila del producto es responsabilidad del cliente, pero se abordan de forma diferente.

Mientras que en los predictivos los requisitos se especifican en documentos formales, en los proyectos ágiles son historias de usuario que forman una pila.

En los requisitos formales, se especifican de forma completa y cerrada al inicio, pero las historias de la pila de usuario están vivas y evolucionan con el desarrollo.

En Scrum el cliente explica su visión al equipo y la pila evoluciona con los aportes de todos.

Existen dos pilas de producto:

- Product Backlog, requisitos desde el punto de vista del cliente, lista de funcionalidades o historias de usuario que desea obtener el cliente ordenadas por la prioridad que les da el cliente.

Esta pila nunca se da por cerrada, están en constante evolución, al principio incluye los principales requisitos y va evolucionando según avanza el desarrollo.

La preparación de la pila de producto se denomina "*grooming*" y se hace de forma puntual entre el propietario del producto y el equipo de desarrollo.

La información mínima que incluye cada historia de usuario es:

- Descripción de la funcionalidad
- Prioridad

- Preestimación del esfuerzo necesario
- Observaciones
- Criterio de validación
- Módulo del sistema al que pertenece

- Sprint Backlog, requisitos desde el punto de vista del equipo de desarrollo, lista de tareas en la que descompuso el equipo las historias de usuario.

  Se confecciona en la reunión de planificación de sprint, y lo hace el equipo, indicando para cada tarea el esfuerzo previsto, este esfuerzo se calcula mediante técnicas como la estimación de póquer.

Ilustración 2.21. Estimación de póker (Alexander Menzinsky, 2016)

  Sólo el equipo podrá modificar esta pila durante el sprint.

  Se deben descomponer las tareas en otras más pequeñas, y nunca una tarea debe llevar más de un sprint de trabajo.

  Se debe poner la pila de tareas en un lugar visible por todos los miembros del equipo, y debe tener un responsable, un estado y el tiempo de trabajo que resta para ser completada. Estos datos serán los que alimentarán el gráfico de avance "*burn-down*".

- Incremento

  Es la parte del producto producida en un sprint, y tiene como característica el estar completamente terminada y operativa para ser entregada al cliente. Se debe producir un incremento en cada Sprint o iteración.

Es habitual tener un Sprint 0, en el inicio del proyecto, para trabajos de investigación previa, diseño o desarrollo de prototipos iniciales.

2.2.3.3 Eventos

- Planificación del sprint

  Se parte del backlog y se determinan cuáles y cómo van a ser las funcionalidades que se incorporarán al producto en el siguiente sprint.

  La organiza el Scrum Máster, y debe asistir el propietario del producto y el equipo completo. Son reuniones largas, en función de las funcionalidades que se deseen incluir.

  Precondiciones:

  - La organización ya tiene formado el equipo que llevará a cabo el sprint
  - Ya están preparadas las historias de usuario y priorizadas
  - El equipo tiene la formación necesaria para llevar a cabo el proyecto

  Entradas

  - Pila del producto
  - Producto desarrollado en los incrementos anteriores
  - KPI de la velocidad de desarrollo del equipo
  - Circunstancias ambientales, estado del negocio, período temporal, etc.

  Salidas

  - Pila de sprint
  - Duración del sprint y fecha de reunión de revisión
  - Objetivo del sprint

  La reunión se divide en dos partes:

  1ª. Que se entregará al final de sprint.

El propietario presenta la pila del producto, expone las historias de usuario de mayor prioridad, si ha habido modificaciones en esta pila, las deberá justificar.

El equipo formulará las preguntas necesarias hasta que sea totalmente consciente del trabajo a realizar. Puede proponer mejoras, modificaciones y soluciones alternativas.

El equipo sintetiza cuál es el valor que se le va a entregar al cliente en este sprint. Esto es una garantía de que todo el equipo comprende y comparte la finalidad del trabajo.

2ª. Cómo se conseguirá hacer el incremento

El equipo desglosa las historias en tareas y estima el esfuerzo para cada una de ellas, componiendo la pila de sprint.

Los miembros del equipo se autoasignan las tareas con una distribución homogénea del trabajo.

El papel del propietario del producto, en esta segunda parte es simplemente atender las dudas que puedan surgir al equipo y asegurarse de que han comprendido bien el trabajo.

- Sprint

  El periodo de sprint es de una duración máxima de 4 semanas, el incremento debe estar completamente operativo y condiciones de ser desplegado.

- Scrum diario

  Es una reunión diaria breve, unos veinte minutos. Se realiza de pie, con todo el equipo presente, cada miembro da respuesta a:

  - En que ha estado trabajando
  - En que va a trabajar
  - Dificultades que está teniendo

  Entradas

- Pila del sprint y gráfico burn-down
- Información de avance de cada miembro

Salidas

- Pila del sprint y burn-down actualizados
- Identificación de posibles necesidades e impedimentos

- Revisión de sprint

  Es la reunión realizada al final del sprint, no debe durar más de 4 horas. Los objetivos son que el propietario del producto compruebe el progreso del sistema, que se identifiquen las historias que ya se puedan considerar hechas y las que no, feedback que permita revisar la pila del producto.

  El equipo no puede invertir más de una hora en preparar la reunión, no se usarán presentaciones ni gráficas, el equipo presenta el resultado final al cliente y responde a sus dudas.

- Retrospectiva

  Después de la reunión de revisión de sprint se debe hacer otra, de unas tres horas, para que el equipo haga un autoanálisis de su forma de trabajar, con el objetivo de corregir posibles malas prácticas y mejorar su productividad y rendimiento. Se debe centrar en "*CÓMO*" se está construyendo.

Las reglas de Scrum se pueden resumir en la siguiente ilustración:

Ilustración 2.22. Marco Scrum técnico (Alexander Menzinsky, 2016)

2.2.4 Estimación ágil

Siguiendo esta metodología cuantas menos medidas mejor, porque medir es costoso, añade burocracia, etc.

Se suelen emplear dos métodos:

- Incremento iterativo: emplea el sprint como unidad de tiempo y define la velocidad como la cantidad de trabajo realizada en ese sprint.
- Incremento continuo: Las unidades de tiempo son días, semanas o meses, la velocidad se expresa en puntos por semana, día o mes.

La gestión ágil no mide el avance por el trabajo realizado, sino por el pendiente de realizar.

No se puede estimar con precisión, ni la cantidad de trabajo de un requisito, ni el tiempo necesario para llevarla a cabo, por esto se sigue la estrategia.

- Trabajar con estimaciones aproximadas
- Estimar con la técnica "juicio de expertos"
- Descomponer las tareas en subtareas más pequeñas que sean más fáciles de estimar.

2.2.4.1 Gráfico de producto "Burn Up"

Es una herramienta de planificación del propietario, proyecta en el tiempo su construcción, en base a la velocidad del equipo.

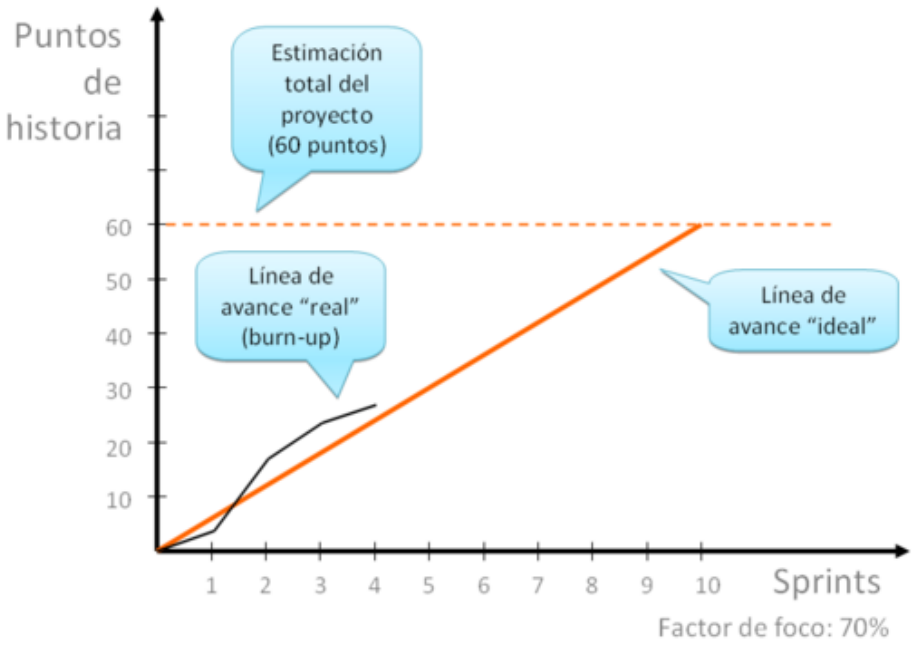


Ilustración 2.23. Gráfico Burn Up (Alexander Menzinsky, 2016)

2.2.4.2 Gráfico de avance "Burn Down"

Lo actualiza diariamente el equipo, midiendo el trabajo que queda por realizar.

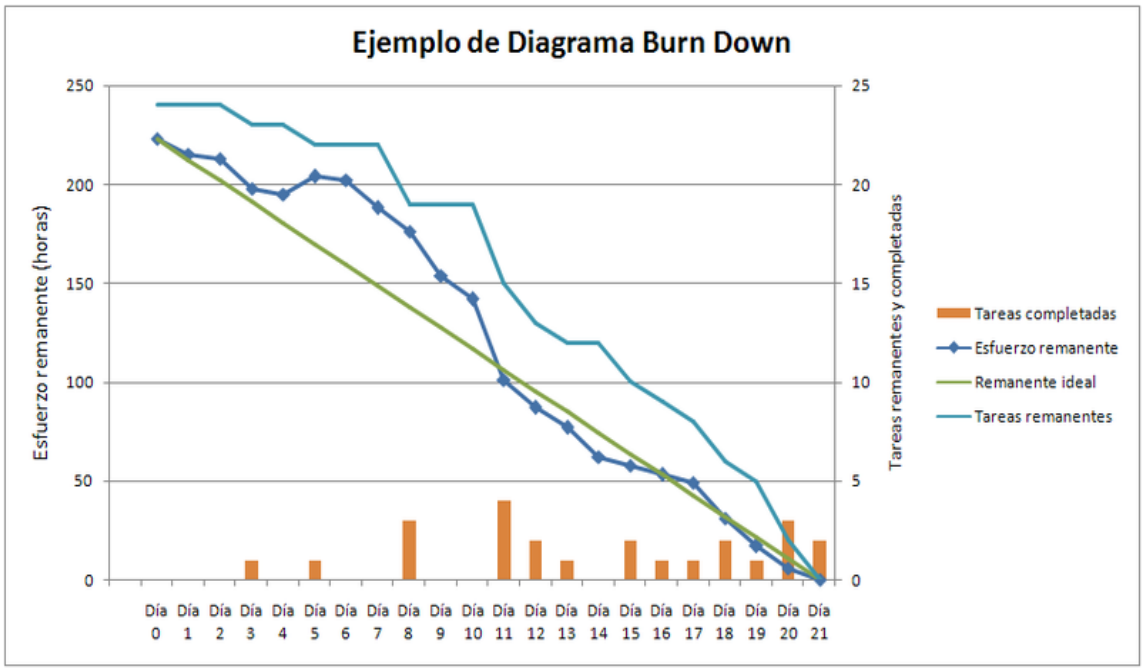


Ilustración 2.24. Gráfico Burn Down (Alexander Menzinsky, 2016)

2.3 **Comparación entre metodologías**

La diferencia fundamental entre estas metodologías se basa en que mientras las tradicionales están enfocadas a la gestión del proyecto, las metodologías ágiles son enfoques de desarrollo de software, que no se preguntan temas más amplios como si el proyecto merece la pena, o si habrá beneficios al final, se centran en las entregas de la manera más eficiente posible e involucran a los clientes en el ciclo constante de definición, priorización, prueba y retroalimentación (Buehring, 2019). Enfoques basados en que "*el proceso de desarrollo es previsiblemente impredecible"*, por lo que se fomenta la transparencia total, la colaboración estrecha y la entrega frecuente de subproductos.

Una de las claves del manifiesto Ágil, es valorar a las personas por encima de los procesos. Esto va en contra de las metodologías tradicionales que creen que con unos buenos procesos se pueden conseguir buenos resultados con personas mediocres. Esto no puede ser cierto en desarrollos de los que se necesita ingenio, creatividad e innovación.

Esta innovación y creatividad se ve reforzada cuando en las primeras fases del desarrollo, durante la fase de especificación, se cuenta con un prototipo sobre el que poder visualizar como podría ser el resultado final. Por contra, la documentación exhaustiva no hace sino atar un producto final mermando la capacidad del desarrollador de mejorarlo e innovar sobre él.

Mientras que el control sobre el proyecto en un método tradicional se vuelca en comprobar que se cumplen los requisitos, el alcance del proyecto, los costos y la planificación temporal, en una metodología ágil, el control se vuelca en proporcionar de forma continua el mayor valor posible al producto. Manteniendo una relación de colaboración continua con el cliente para poder llevarla a cabo.

Los principales valores de la gestión ágil son la anticipación y la adaptación, mientras que en la ortodoxa son la planificación y control.

En resumen, para poder desarrollar un proyecto de forma predictiva, se necesita una definición muy clara de lo que se pretende con dicho proyecto. Una especificación de requisitos exhaustiva y tener todo muy bien documentado y firmado por parte del cliente. Esto hará que el equipo de desarrollo trabaje intentando cumplir con las especificaciones

en el plazo previsto y sin necesidad de tomar decisiones que afecten a cambios de funcionalidad o mejoras del propio proyecto. Es posible realizar una gestión del tiempo y unos presupuestos precisos que se cumplen con pocas desviaciones y esto satisface a los clientes. Pero una alteración del proyecto en una etapa avanzada ocasiona graves problemas de costes y tiempos.

Sin embargo, en metodologías agiles, no es necesario unas especificaciones tan concisas ya que el equipo se adapta a las necesidades requeridas, generando valor al cliente en plazos cortos de tiempo y permitiendo que este pueda variar los requisitos o las funcionalidades con frecuencia para obtener, al final, el proyecto que realmente necesita.

Para poder realizar este tipo de metodologías es necesario contar con un buen equipo que sea capaz de autoorganizarse, que esté involucrado con la empresa, con experiencia previa en este tipo de proyectos y a poder ser con los mismos miembros de equipo desde el inicio hasta el final del proyecto. Es vital la comunicación entre los involucrados, es por ello que se debe llevar una buena gestión en este punto. La falta de planificación del proyecto, desde los plazos de entrega hasta los presupuestos, dan la sensación de que todo se define sobre la marcha en una estructura débil.

Prince2	Scrum
Sirve para que el cliente justifique el proyecto	Sirve para que el proveedor entregue el software
Se centra en los niveles de dirección superiores	Se centra en los equipos de nivel inferior
Responde a: ¿deberíamos hacer el proyecto? ¿merece la pena en cuanto a coste y riesgo?	Responde a: ¿qué entregamos la semana que viene? ¿Cómo sabremos que un producto está terminado?
Enfoque más predictivo	Enfoque más adaptativo

Tabla 2.4 Diferencias principales Prince2 & Scrum (autor)

Sin embargo, autores como (Suárez, 2010) establecen que sería posible hacer un hibrido entre metodologías predictivas y ágiles, y que de hecho, utilizar PRINCE2 junto con SCRUM mejoraría las probabilidades de éxito en la gestión del proyecto. Prince 2 se encarga de llevar el control y la estabilidad del proyecto a nivel general, mientras que Scrum lleva el control y la agilidad a bajo nivel.

Cuando se combinan estas metodologías, Prince2 se ocupa de gestionar el control y la estabilidad en un nivel general del proyecto, mientras que la metodología ágil de gestionar el control y la agilidad del proyecto a más bajo nivel de detalle del proyecto.

De hecho, el principal concepto de Scrum de trabajar en Sprints o time-boxes, encaja perfectamente con el concepto de Prince2, de un plan de equipo, donde Prince2 contempla la posibilidad de que haya más de un time-boxes dentro del plan del equipo.

Prince2 tampoco aborda como debe de organizarse un equipo en un proyecto, considera que es mejor que los propios equipos seleccionen la mejor forma de trabajar. Esto es además uno de los principios ágiles por lo que encaja fácilmente en un proyecto Prince2.

Existe una percepción errónea de que Prince2 tiene un enfoque de "cascada", donde se documentan y aprueban los requisitos antes de pasar de diseño a construcción y finalmente a pruebas. Pero esto no es así como se puede ver en el último manual de Prince2 (2017), donde se asume que las necesidades del proyecto surgen y evolucionan durante la vida del proyecto. Estos cambios los gestiona en el control de cambios, sin embargo, los cambios de nivel inferior, como las solicitudes de funciones, pueden gestionarse a nivel de equipo como pasa en las técnicas ágiles, sin que sea necesario escalar estas decisiones, como se vio en la gestión por excepción.

La utilización de enfoques ágiles en proyectos Prince2 puede aportar lo mejor de ambos mundos: la estructura y dirección de Prince2, junto con la flexibilidad y capacidad de respuesta de Agile.

Prince2 define una interfaz sencilla entre la organización del cliente y la organización del proveedor, esto implica que los equipos de un proyecto Prince2, pueden utilizar cualquier enfoque de desarrollo que elijan, incluyendo cualquiera de los enfoques Agile, por lo que siempre que cumplan con la interfaz definida por Prince2, los equipos pueden utilizar los beneficios de Agile, mientras que el cliente mantiene los beneficios de Prince2 centrados en la justificación del negocio.

2.4 **Conclusión**

La simple elección de un método o enfoque nunca garantizará el éxito de un proyecto, cualquier método o enfoque utilizado de manera estúpida hará un desastre de cualquier proyecto.

Aunque el método predictivo a priori es un método ideal, desde el punto de vista de que se definen las funcionalidades y requisitos en la fase inicial y después se trata de controlar el desarrollo, riesgos, gestión de plazos, gestión del conocimiento, gestión documental, gestión de involucrados, gestión de recursos y cerrar el proyecto. Se puede ver, como indica el informe Chaos (Group S. , 2015), que la definición inicial, en un porcentaje alto, no es correcta, bien porque no llega correctamente el mensaje al equipo de desarrollo final o bien porque el cliente, en esa fase, todavía no tiene claro la idea final de lo que necesita.

Esto deriva a que todo el control posterior se haga bajo unas premisas erróneas, dando lugar a resultados, como vimos, no esperados por el cliente. Este resultado estará muy atado contractualmente en la fase inicial, por lo que la insatisfacción del cliente no es traducida en una pérdida económica para el suministrador, pero si en una pérdida de prestigio y en la mayoría de los casos en la pérdida de oportunidad de realizar nuevos proyectos con dicho cliente.

Por otro lado, en cuanto a la gestión de equipos, se vuelve muy aristocrática ya que el equipo parte de una definición estricta y simplemente la tiene que desarrollar, coartando su creatividad y propuestas de mejoras, que puedan observarse durante los propios desarrollos.

Como definieron Robert Tannembaum y Warrem Schmidt (Schmidt, 1973) el director de proyecto adapta su estilo de liderazgo a las circunstancias del proyecto y del equipo.

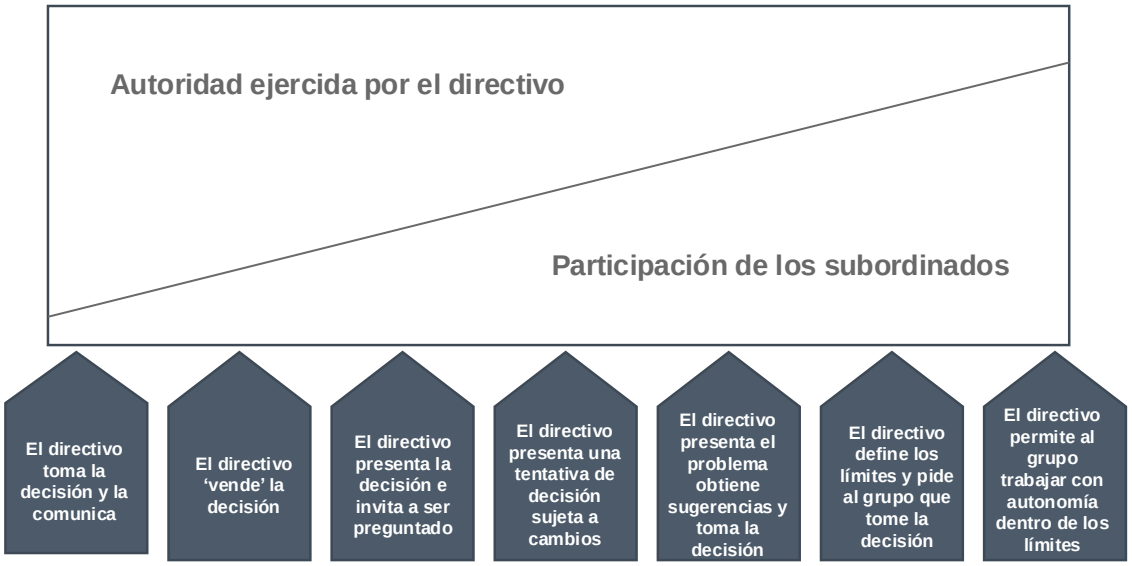


Ilustración 2.25. Como elegir un patrón de liderazgo (Schmidt, 1973)

Como se ve en la ilustración 28, en un proyecto donde todo queda perfectamente definido en la primera fase por parte del director del proyecto y el cliente, no queda nada más que comunicar al equipo de desarrollo lo que tienen que hacer, por lo que se podría avanzar únicamente hasta un liderazgo en el que el jefe de proyecto presente la decisión ya tomada e invite al grupo a ser preguntado, pero nunca va a ser una decisión expuesta a cambios. Por este motivo, la motivación del equipo, el entusiasmo y la involucración de este será menor. Otra vez se va a la premisa inicial de que, con unos buenos procedimientos, no es necesario un gran equipo.

Por otro lado, en las metodologías ágiles tipo Scrum, podemos ver el caso contrario, en estas el estilo de liderazgo por circunstancias de la metodología vemos que llega al punto en el que el director del proyecto permite al grupo trabajar con autonomía dentro de los límites.

Estos factores fueron analizados en el modelo de liderazgo participativo de Vroom y Jago (Yetton, 1988), comprobando que este modelo tiene ventajas en cuanto a compromiso de los trabajadores con la organización, mayor motivación para el alcance de los objetivos asignados, y mayor integración del factor humano en la organización. Incrementa la aptitud positiva de los trabajadores hacia sus dirigentes y la satisfacción en el trabajo. Lo que lleva a buenos ambientes de trabajo.

Esto se da en trabajadores con necesidad de independencia, e igualitarias en cuanto a creencias y valores. Por el contrario, las personas autoritarias no incrementan mayor grado de satisfacción en estos ambientes de participación.

La calidad de las decisiones, entendiendo decisión de calidad por aquella que, de llevarse a cabo, logrará alcanzar los objetivos de la organización, se ve también afectada en estos ambientes participativos, y esto va a depender de los objetivos de los participantes, sus conocimientos, el tamaño del grupo, los desacuerdos de los participantes y la naturaleza del problema. Por eso un ambiente democrático no necesariamente va a ser bueno para la organización y es algo que se debe tener en cuenta en la metodología de trabajo a seguir.

Por otro lado, la participación contribuye al desarrollo personal de los trabajadores y las relaciones interpersonales del grupo. Contribuye a formar equipos más robustos y con mayor confianza.

Aunque es muy motivante para los equipos de desarrollo ya que se les da pie a opinar, crear y decidir, según el continuo de Tannembaum este carácter democrático en los equipos sólo funciona con equipos muy maduros y durante períodos de tiempo cortos, por lo que en empresas de outsourcing donde el perfil de programadores es joven, inexperto y muy cambiante, es difícil de llevar a cabo en todas sus vertientes.

Prince2 ayuda a asegurar que los proyectos tengan un sentido comercial sólido, y Agile tiene la capacidad de responder rápidamente a los cambios con la entrega a tiempo de los productos.

Entonces, ¿Por qué elegir entre uno u otro? Los proyectos pueden beneficiarse de utilizar el enfoque predictivo de Prince2 junto con el adaptativo de los enfoques ágiles, de este modo, se entregan los productos respondiendo plenamente al entorno cambiante a la vez que se garantiza que la inversión ha sido acertada.

De su actualización en 2017, el manual de Prince2, ya contempla su utilización junto con metodologías ágiles.

A partir de este estado del arte se deduce que actualmente no hay una metodología reconocida que se adapte a las necesidades de las empresas de outsourcing, con equipos de trabajo formados en su mayor parte por gente junior, y donde los requerimientos definidos por parte de los clientes en la industria son en la mayor parte, definiciones

pobres. Pero sí que se empieza a considerar por parte de las principales metodologías como Prince2, la gestión más adaptativa y ligera en cuanto a burocracia, y documentación que no aporta valor a los proyectos.

Es necesario una metodología capaz de resolver:

- ✓ **Expresión de necesidades**: donde se recojan las necesidades por parte de los ingenieros y del proceso, pero también, los requisitos y puntos de vista de los operarios que tendrán que trabajar con ello.
- ✓ **Especificaciones**: Un procedimiento para llevar a cabo una especificación a partir de los datos anteriores y que cumpla con los estándares de calidad de la compañía, requisitos de seguridad, sea legible, tenga en cuenta las pruebas que deberá pasar el proyecto quedando también definidas en este documento, y describa las características de los entregables finales.
- ✓ **Análisis y diseño**: Como se llevará a cabo este análisis, que entregables deberá generar, donde se almacenará está documentación, como se gestionaran los posibles riesgos, como se gestionará el conocimiento adquirido.
- ✓ **Desarrollo y testeo**: Como se gestionará el equipo de desarrollo, que procedimientos se seguirán para dar por terminada una tarea, como se validará, que métricas se usaran para medir la velocidad de desarrollo y la calidad del código.
- ✓ **Evolución**: Cuales son las claves para la realización de cambios y la inclusión de mejoras en el proceso, que fases deberán cumplir, en qué casos estas evoluciones deberán tratarse como un nuevo proyecto, y en qué casos como una continuación del existente.
- ✓ **Cierre de proyecto**: Procedimiento para cerrar un proyecto. Definición de los entregables mínimos que habrá que aportar.

3 Problemática de los proyectos de outsourcing para la industria y encaje de las metodologías de "Project management"

Estamos ante la cuarta revolución industrial o industria 4.0, las empresas del sector están convirtiendo sus fábricas en *Smart Factorys* por lo que el desarrollo software y la automatización de procesos es su principal inversión.

El ciclo de vida del software indica el desarrollo software desde que se detecta la necesidad, hasta que se implanta el producto, pasando por todas las etapas intermedias de desarrollo, testeo y evolución que garantizan que el producto final cumpla con las especificaciones pedidas. Existen varios modelos de ciclo de vida, en este trabajo se usa el indicado en la siguiente ilustración.

Ilustración 3.1. Ciclo del software (autor)

A lo largo de este epígrafe se describe la forma más habitual de trabajo desde el punto de vista del ciclo del software, en los desarrollos de outsourcing para la industria. Se analizan todas las fases desde que a alguien se le ocurre una mejora, o tiene una necesidad, a como esta idea se convierte en proyecto, se realiza su especificación a una empresa externa de desarrollo de software, que la desarrolla y finalmente se implanta y evoluciona.

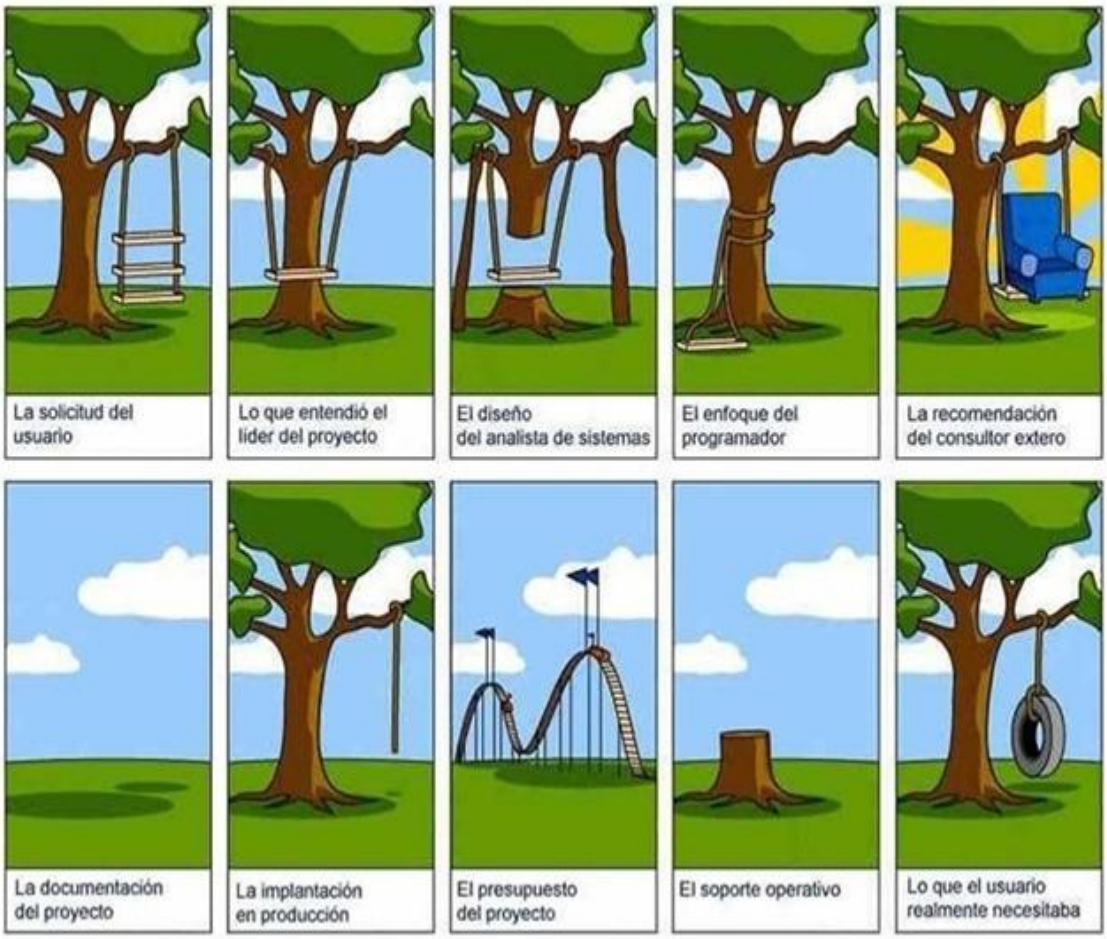


Ilustración 3.2. La realidad ilustrada (ciclo del software)

De este modo, se analiza la forma actual de trabajar, y se compara de cómo se haría si se siguiese la metodología Prince2, qué procesos se aplicarían en cada fase y lo mismo si se usase Scrum.

Se analizan los problemas que surgen en el día a día de la gestión de este tipo de proyectos, ahondando en la causa del por qué se dan esos problemas, se analiza como serían esos problemas gestionando el proyecto con otras metodologías y finalmente se propone una solución.

No todas las fases en Prince 2 o Scrum son extrapolables a las fases dentro del este ciclo de vida del software, para poder hacer este análisis, se han considerado los procesos principales.

3.1 **Expresión de necesidades**

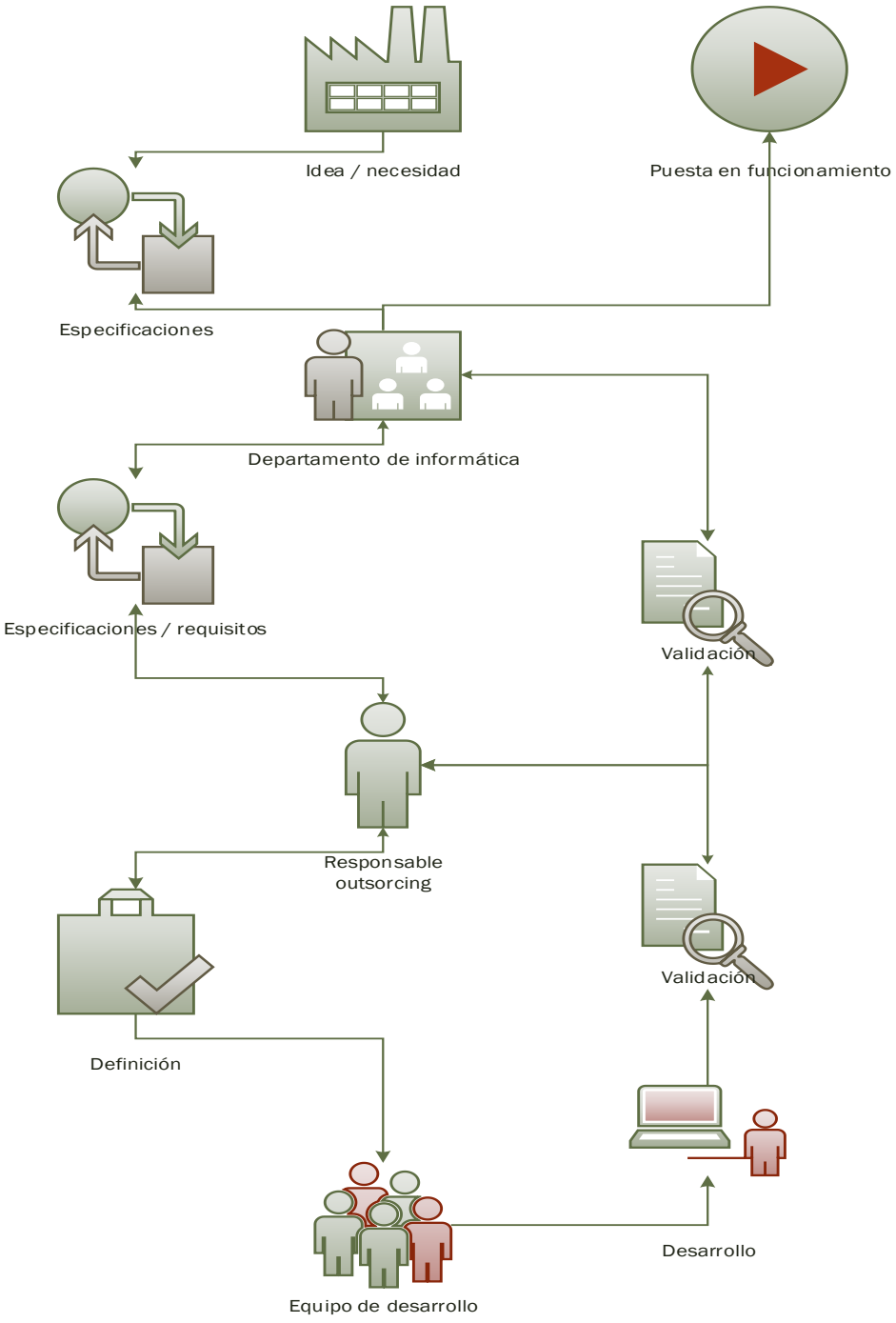


Ilustración 3.3. Expresión de necesidades (autor)

En la industria, se detecta una necesidad en el proceso de fabricación. Dicha necesidad es detectada por el departamento de producción que por lo general no son expertos en software, son profesionales del mundo de la ingeniería u operarios.

Dicha necesidad, es comunicada al departamento de informática de la propia empresa, que desarrollará una definición de esa idea, y dicha definición será discutida con el responsable de la empresa de outsourcing, que le aportará valor si cabe, y recogerá las especificaciones y requisitos para más tarde en su empresa, desarrollarla.

3.1.1 Principales problemas detectados

Falta de comunicación directa entre el usuario final y el desarrollador. Incluso en muchos casos, el propio ingeniero responsable, no tiene el suficiente conocimiento de lo que realmente necesita el operador o lo que demanda.

Poco conocimiento del desarrollador sobre:

→ El proceso de fabricación y sus principales características.

→ Sobre el entorno físico de trabajo del operador final.

→ Sobre la utilidad de dicho desarrollo.

→ Sobre los riesgos de un mal uso de esta.

→ Sobre otros desarrollos similares ya implantados.

→ Sobre la arquitectura software de la empresa.

3.1.2 Riesgos en esta fase

Por una mala comprensión de la necesidad se pueden perder muchas horas de trabajo de varias personas, dando lugar a un desarrollo con poca o nula eficacia sobre la necesidad real.

Es posible que no se llegue al resultado esperado por parte del cliente, y que el operador nunca llegue a trasladar que ese producto no es el producto que necesita.

Un ejemplo real sobre esto es una pantalla solicitada, que mostraba un gráfico de temperaturas. Al final se descubrió que, de la gráfica mostrada, con todos sus vértices, y precisión, el operador, para tomar una decisión dentro del proceso, lo único que necesitaba era saber cuándo la gráfica sobrepasaba un límite de temperatura, que además

era un límite fijo en todos los casos. Por lo que el operador, en vez de solicitar que le modificasen la pantalla, dibujó directamente una línea recta con un rotulador en dicho limite, sobre el propio monitor.

El resultado podría haber sido mucho mejor, abarcando muchos más campos y con más precisión, de haber habido una mejor comprensión de la necesidad real y una mejor recogida de feedback.

Otro ejemplo real sobre esto se dio al migrar una pantalla que contenía un grid que mostraba la información de las piezas que había en un horno.

A pesar de no venir en la especificación, el equipo de desarrollo decidió que era interesante poner en algún lugar de la pantalla, el número total de piezas que estaban en ese momento dentro del horno.

Al implantar la pantalla, esta vez sí, con recogida de feedback, de todas las mejoras realizadas en el diseño de la nueva pantalla, los operadores se mostraban encantados por la inclusión del nuevo campo que ya no les obligaba, a cada poco, tener que contar las filas de la tabla (unas 70), para comprobar cuantas piezas estaban dentro.

Son algunos de los ejemplos de porque es tan importantes esta fase, y una fase posterior, de recogida de feedback y auditoria de calidad.

3.1.3 Tratamiento en Prince 2

En esta fase todavía no es un proyecto, es una necesidad y se deberá decidir si se convierte en proyecto, pero esto no corre a cargo de la empresa externa decidirlo, por lo que a estas reuniones sólo acuden gente de la propia empresa, y en raras ocasiones el consultor externo para dar su opinión como especialista en la tecnología, nunca para tomar una decisión.

Una vez que esa necesidad se comunica a la empresa de outsourcing, siguiendo Prince 2, los procesos que comienzan tendrían que:

- Autorizar la preparación de un plan de proyecto.
- Comprobar que el proyecto es justificable.
- Autorizar la puesta en marcha y definir quién va a ser el equipo responsable de llevarlo a cabo.

- Identificar el tipo de solución que se va a proporcionar.
- Identificar las expectativas de calidad del cliente
- Crear el registro de riesgos conocidos hasta el momento
- Planificar la etapa de iniciación
- Se establece:
    - Lo que hay que hacer
    - Quién toma las decisiones
    - Quién paga el proyecto
    - Quién indica las necesidades
    - Qué estándares de calidad se requieren

3.1.3.1 SU1. Designar la junta de proyecto y el jefe de proyecto.

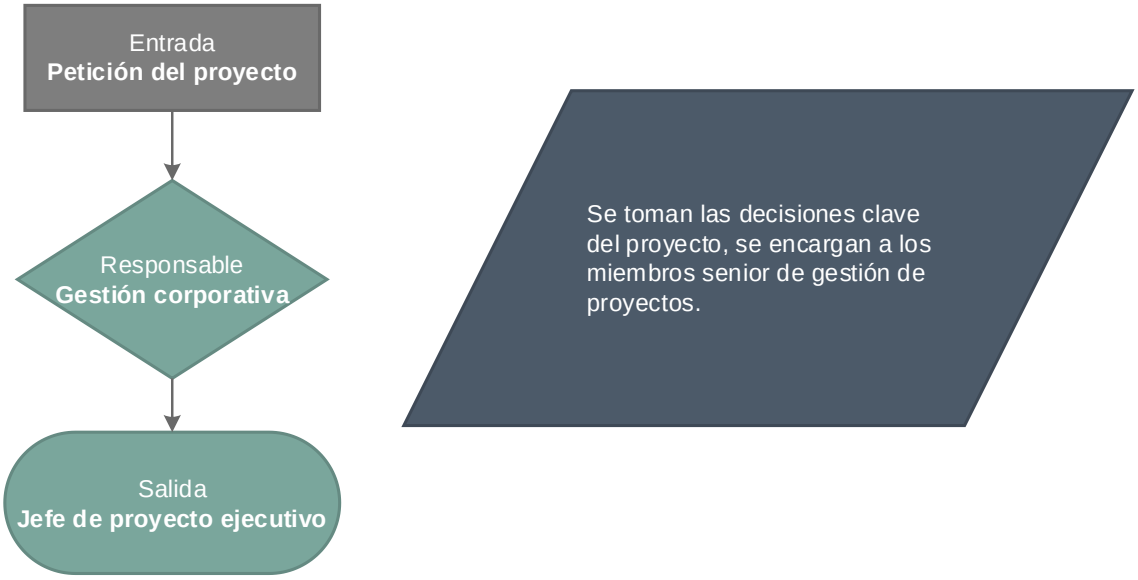


Ilustración 3.4. SU1 Designar la junta de proyecto (autor)

3.1.3.2 SU2. Diseñar un equipo de gestión del proyecto

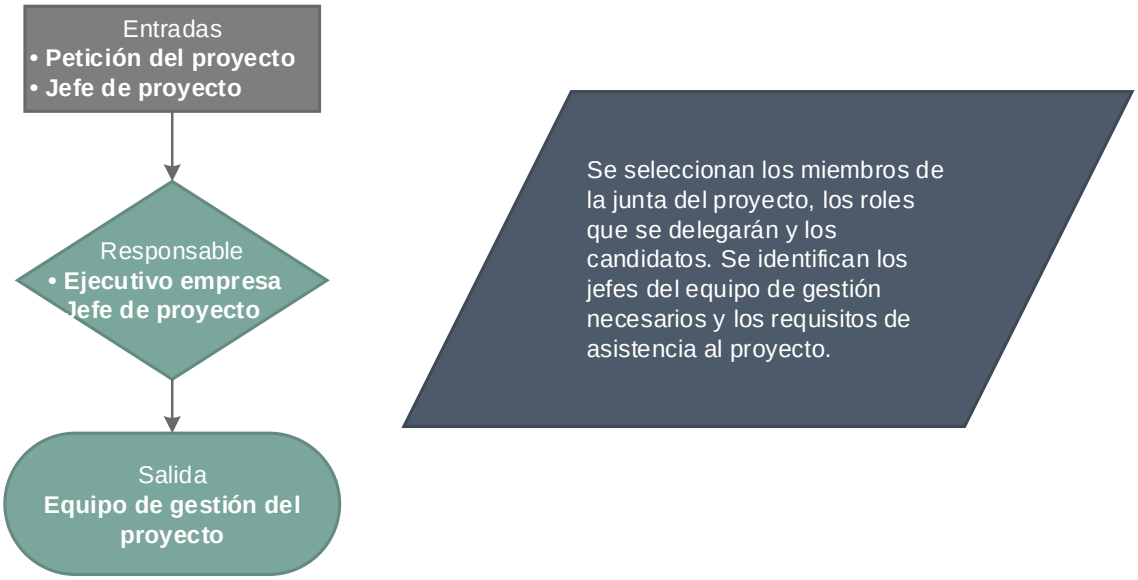


Ilustración 3.5. SU2 Diseñar el equipo de gestión (autor)

3.1.3.3 SU3. Designar un equipo de gestión del proyecto

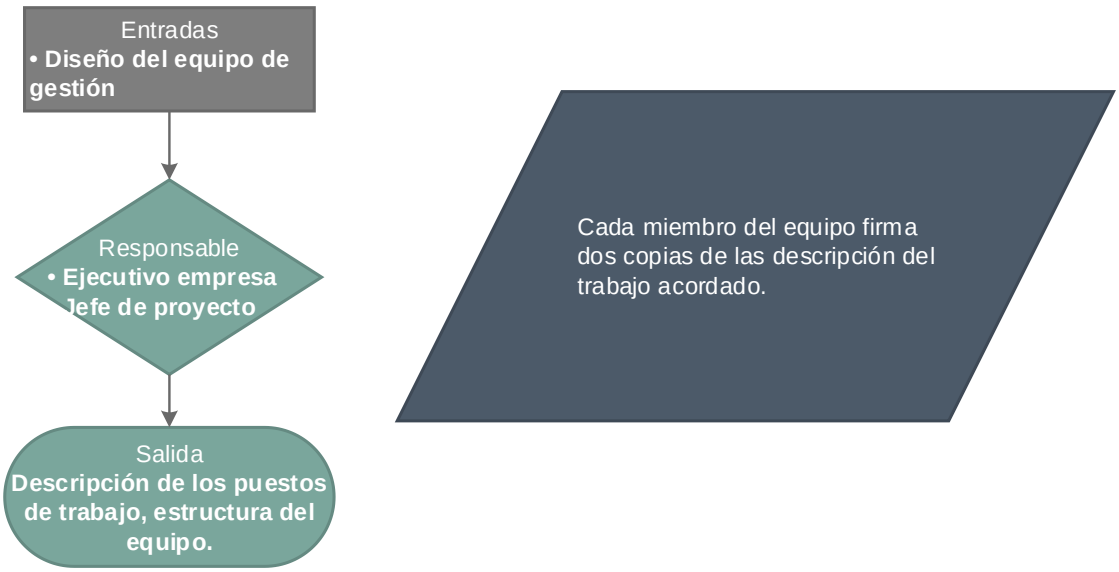


Ilustración 3.6. SU3 Designar un equipo (autor)

3.1.3.4 SU4. Preparar el informe preliminar del proyecto

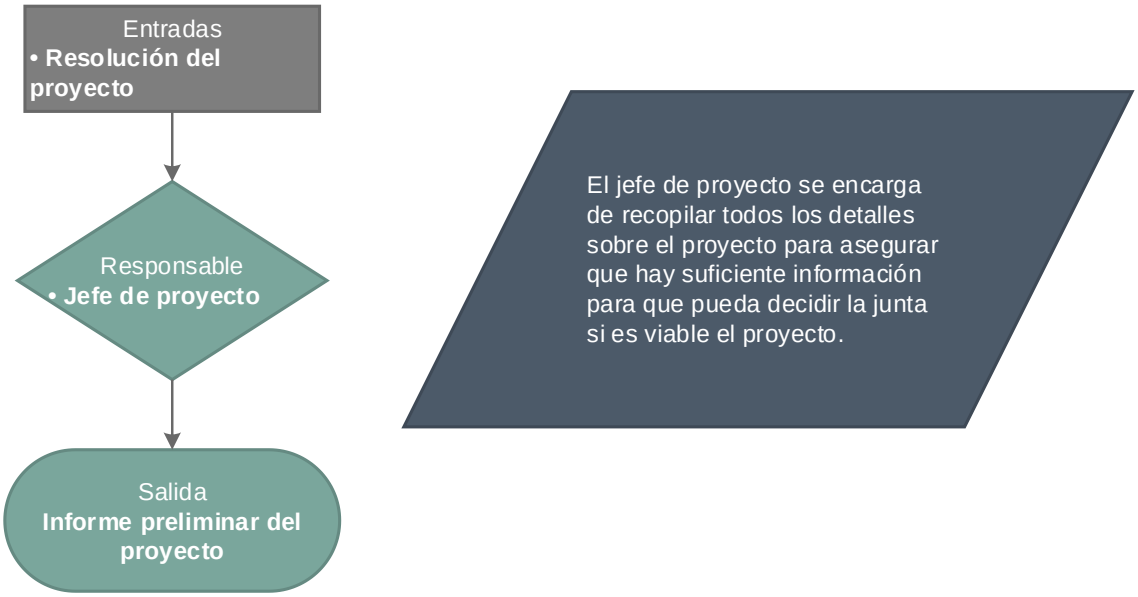


Ilustración 3.7. SU4 Informe preliminar (autor)

3.1.3.5 SU5. Definir la aproximación al proyecto.

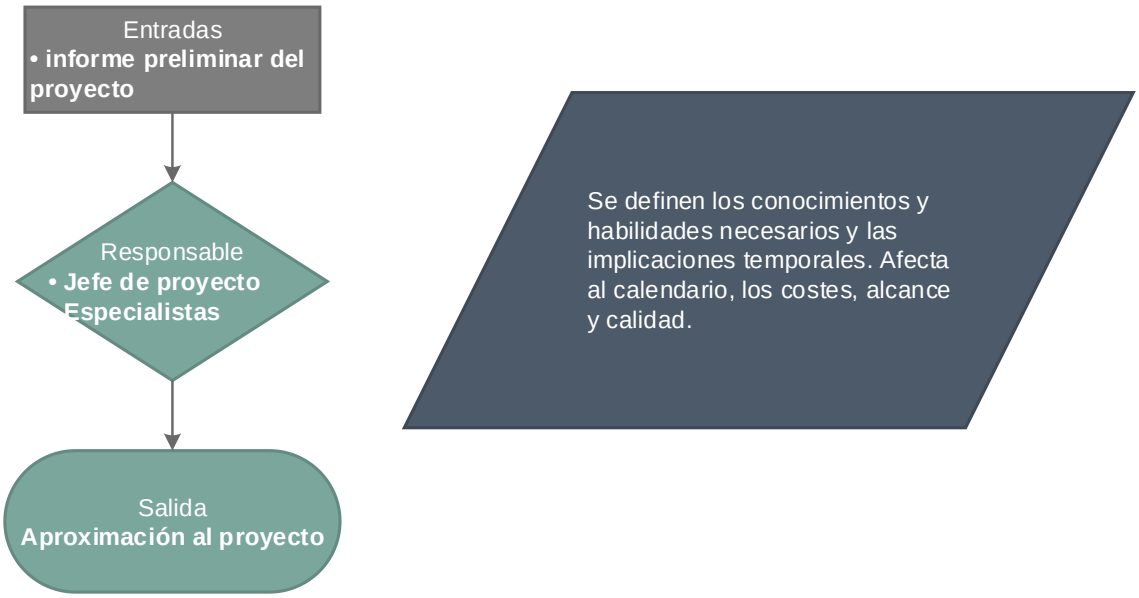


Ilustración 3.8. SU5 Aproximación al proyecto (autor)

3.1.3.6 SU6. Planificar la etapa de iniciación.

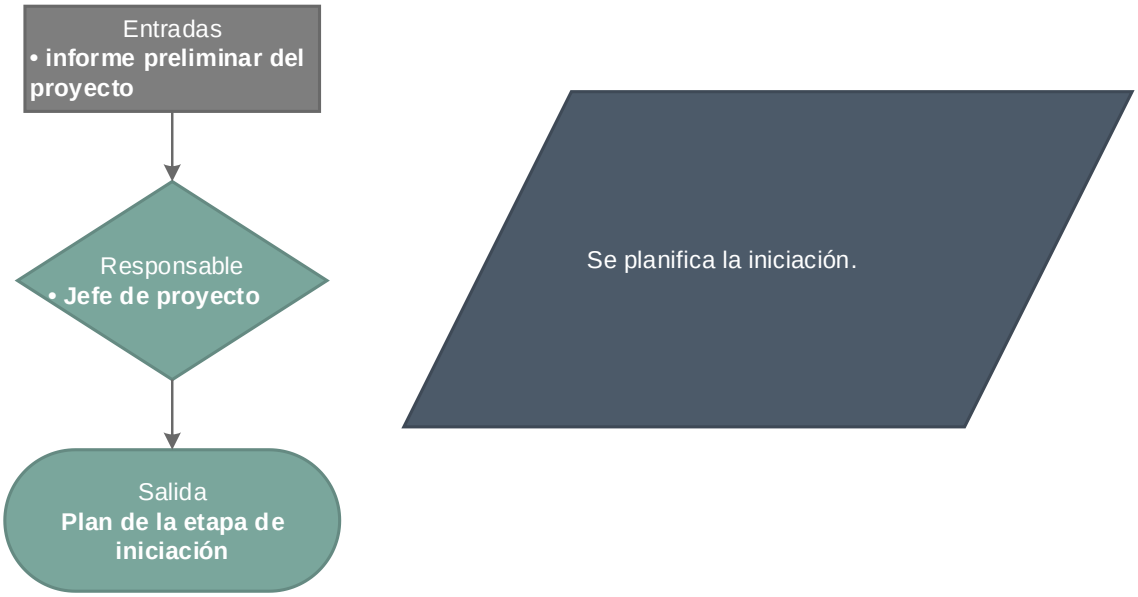


Ilustración 3.9. SU6 Etapa de iniciación (autor)

3.1.4 Tratamiento según Scrum

En Scrum esta parte no se trata.

3.2 **Especificaciones**

Esta es seguramente una de las fases más críticas de todo el ciclo, ya que los errores en esta se arrastrarán durante todo el ciclo, por lo que puede dar lugar a una mala calidad del desarrollo o en el mejor de los casos una perdida sustancial de tiempo y su coste derivado.

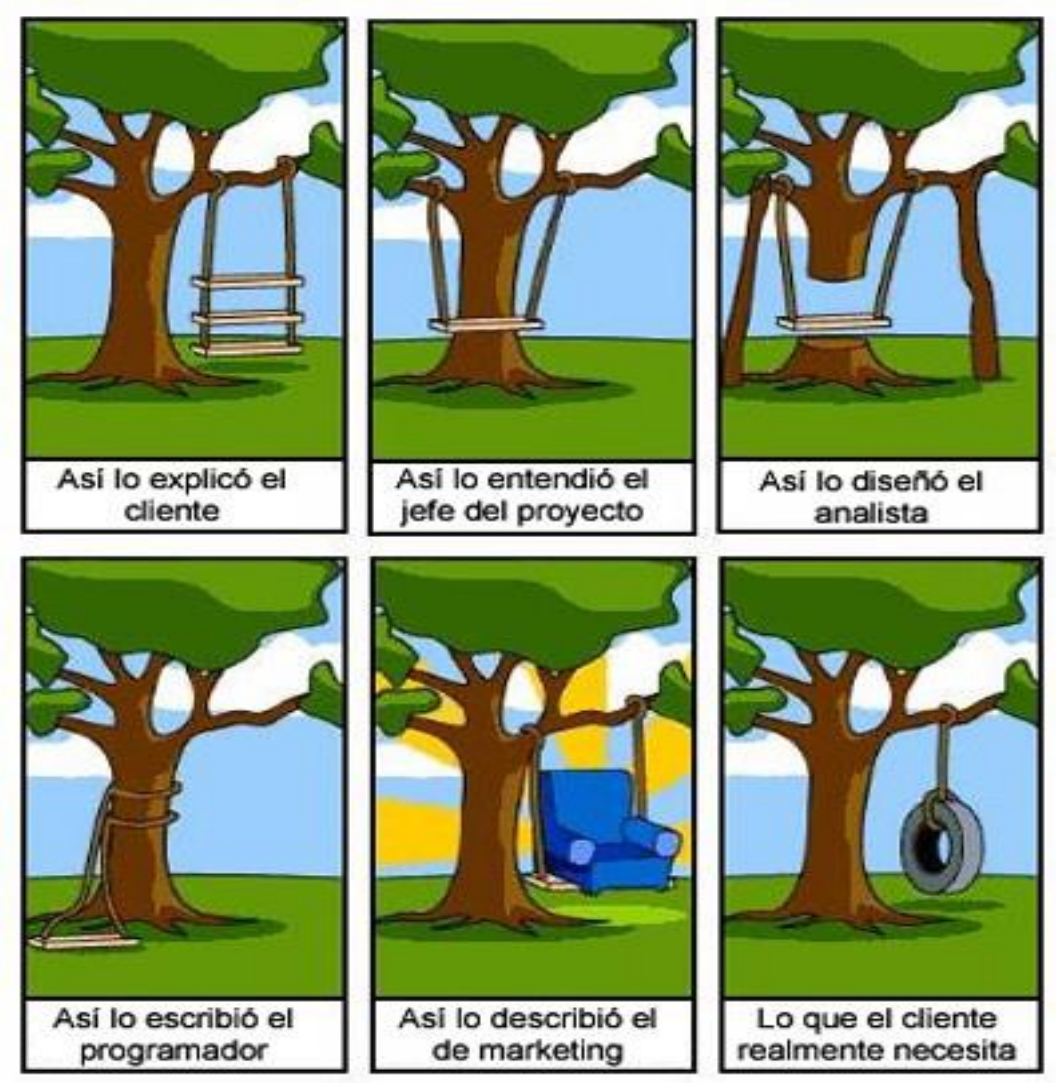


Ilustración 3.10. La realidad ilustrada (gestión de requisitos)

3.2.1 Principales problemas detectados

Habitualmente es el departamento de informática de la compañía, que a su vez es el departamento que tiene externalizados los desarrollos, los que se encargan de realizar la especificación tras una reunión inicial con el cliente final.

Este gestor, conoce el proceso, entiende bien el resultado que se pretende obtener, pero no conoce al equipo de desarrollo, sus virtudes y sus carencias. Además, al estar enfrentado a los problemas diarios del proceso y sus mantenimientos, en la mayoría de los casos no está al tanto de las nuevas innovaciones tecnológicas y de los beneficios que se pueden obtener de las mismas.

Por ello, hará una especificación en base a sus conocimientos del proceso y del resultado que supone espera obtener el usuario final. Son especificaciones que pocas veces son innovadores o que busquen un avance tecnológico, simplemente suelen ser especificaciones que buscan resolver el problema de una forma conocida.

En muchos casos se parte de especificaciones de proyectos similares anteriores, por lo que se arrastran problemas del pasado ya superados en la tecnología actual.

Dicha especificación llegará al responsable de outsourcing que será el encargado de analizarla y proponer al gestor de informática una solución final.

Un ejemplo lo podríamos tener en un proyecto simple, consistente en realizar la comunicación entre una nueva máquina con el servidor que deberá recoger los datos de dicha máquina. Es fácil caer en la tentación de utilizar la misma comunicación que tenía la máquina que se va a sustituir, diseñada e implantada en los 80, con los problemas tecnológicos de esos años, sin pararse a analizar qué problemas tenía dicha máquina, y cuáles son las mejoras que se podrían implantar en esta reforma. Esta decisión puede tomarse simplemente por el hecho de que con los años la vieja comunicación se ha depurado, y ha llegado a estabilizarse generando pocos problemas en la actualidad, es decir, *como funciona, mejor no tocarlo*.

También hay que considerar la documentación generada para esta especificación, en muchos casos decenas de emails, con preguntas y respuestas entre varios usuarios, y de ahí el responsable de outsourcing obtiene un documento que sirve a los desarrolladores para hacer su trabajo, pero que cuando se entrega el producto final y se detectan fallos en la especificación inicial siempre queda la duda de en qué momento se interpretó mal dicho requisito o si al mejor, ni se llegó a especificar por escrito, porque fue algo que simplemente alguien comentó durante una reunión vía Skype.

3.2.2 Riesgos en esta fase

- Al ser especificaciones carentes de innovación, se repiten patrones y diseños ya conocidos y que no mejoran la eficiencia de los existentes.
- No se aprovecha, los avances tecnológicos, ni los conocimientos que puede tener algún miembro del equipo para aportar una mejora sustancial en el proyecto.
- Da lugar a proyectos poco motivantes para el equipo de desarrollo y esta falta de motivación puede acarrear una falta de calidad en el producto final.
- Mala interpretación del proyecto puede generar cuantiosos costes en la etapa de gestión de cambios, si se desarrollan estructuras que luego no sean fáciles de modificar y adaptar a la especificación real.
- En algunos casos se pueden generar errores difíciles de detectar durante el proceso y que queden ya siempre latentes en los proyectos, pudiendo generar cuantiosas pérdidas o incluso poner en riesgo la seguridad y salud de las personas. Un ejemplo de esto sería la mala interpretación de un cálculo estadístico que detecte un defecto en la pieza, al final puede generar un error en el proceso que pase por alto piezas que en realidad estén mal y que puedan ocasionar accidentes en algunos casos, aunque estos defectos y estos accidentes nunca lleguen a poder ser relacionados con la mala especificación y por eso nunca lleguen a corregirse.

3.2.3 Tratamiento en Prince 2

En este punto Prince 2 pone en marcha el proyecto, y para ello:

- Define la responsabilidad, los métodos y herramientas para la calidad.
- Planifica el proyecto entero
- Establece los fundamentos para un proyecto bien planificado y controlado
- Confirma que se trata de un caso de negocio viable
- Reevalúa los riesgos del proyecto.
- Confirma todos los puntos de decisión acordados para el proyecto

3.2.3.1 IP1. Planificar la calidad

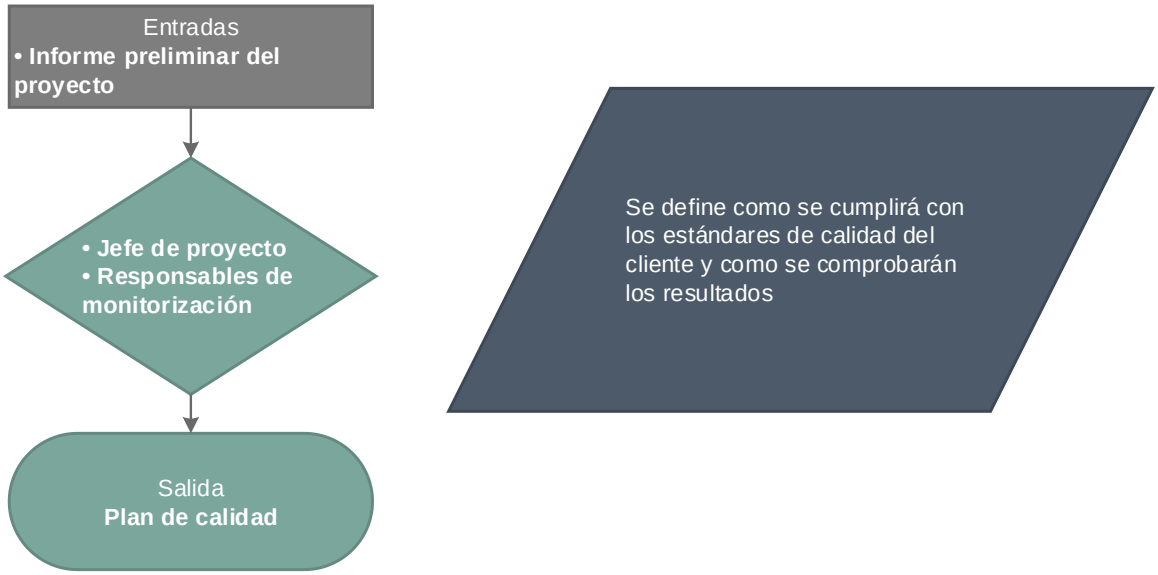


Ilustración 3.11. IP1 Planificar la calidad (autor)

3.2.3.2 IP2. Planificar el proyecto

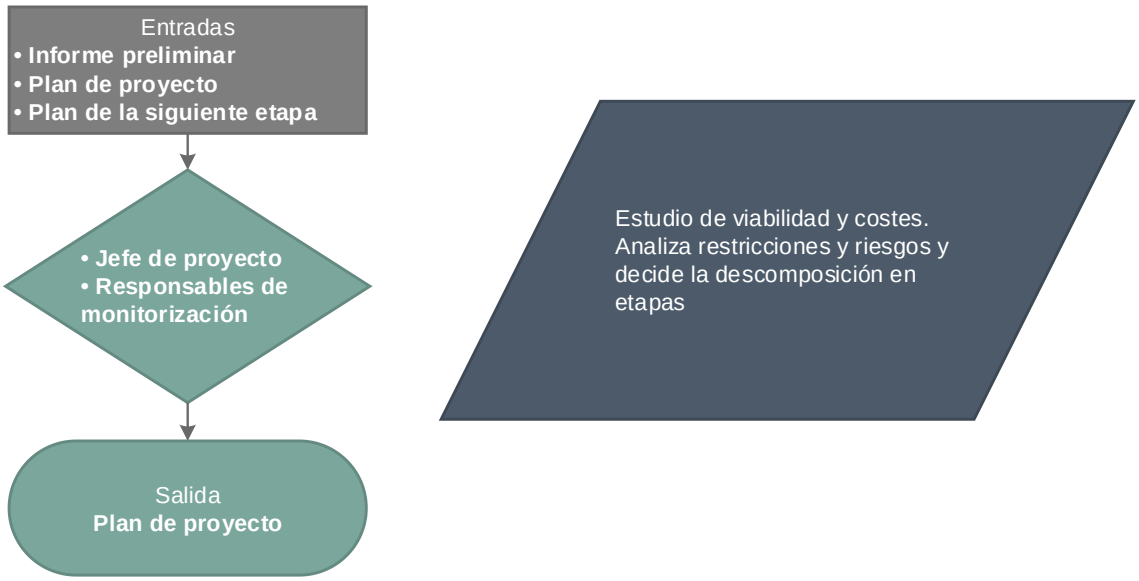


Ilustración 3.12. IP2 Planificar el proyecto (autor)

3.2.3.3 IP3. Refinar el caso de negocio y los riesgos

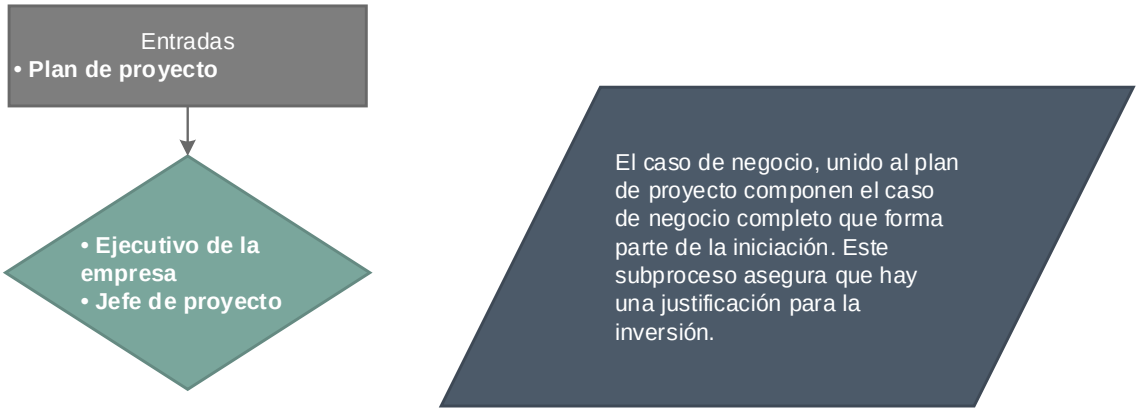


Ilustración 3.13. IP3 Refinar el caso de negocio (autor)

3.2.3.4 IP4. Establecer el control del proyecto

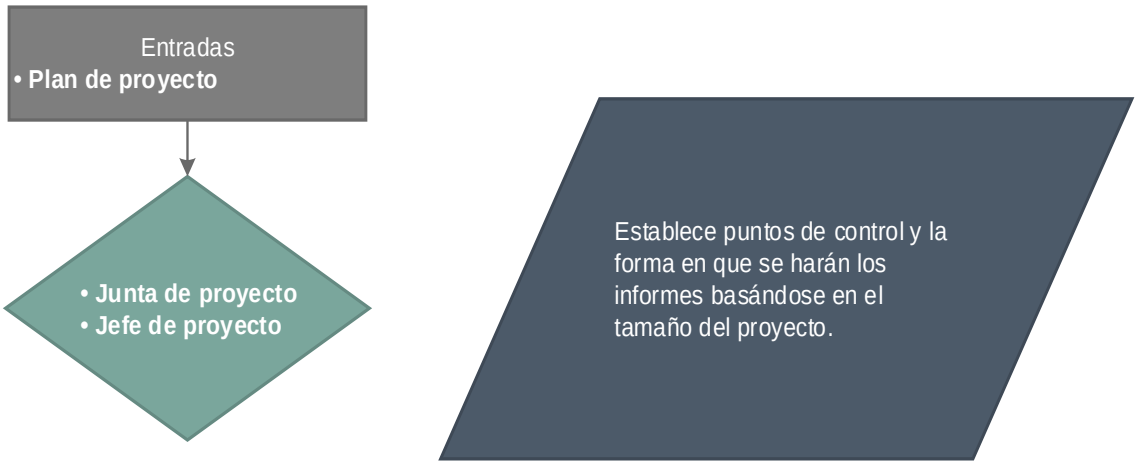


Ilustración 3.14. IP4 Establecer el control del proyecto (autor)

3.2.3.5 IP5. Establecer los archivos del proyecto

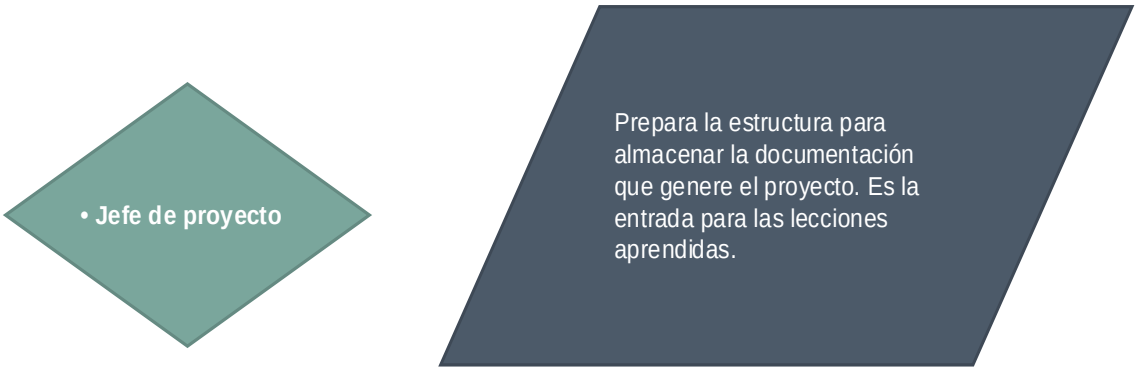


Ilustración 3.15. IP5 Establecer los archivos del proyecto (autor)

3.2.3.6 IP6. Ensamblar el documento de iniciación del proyecto

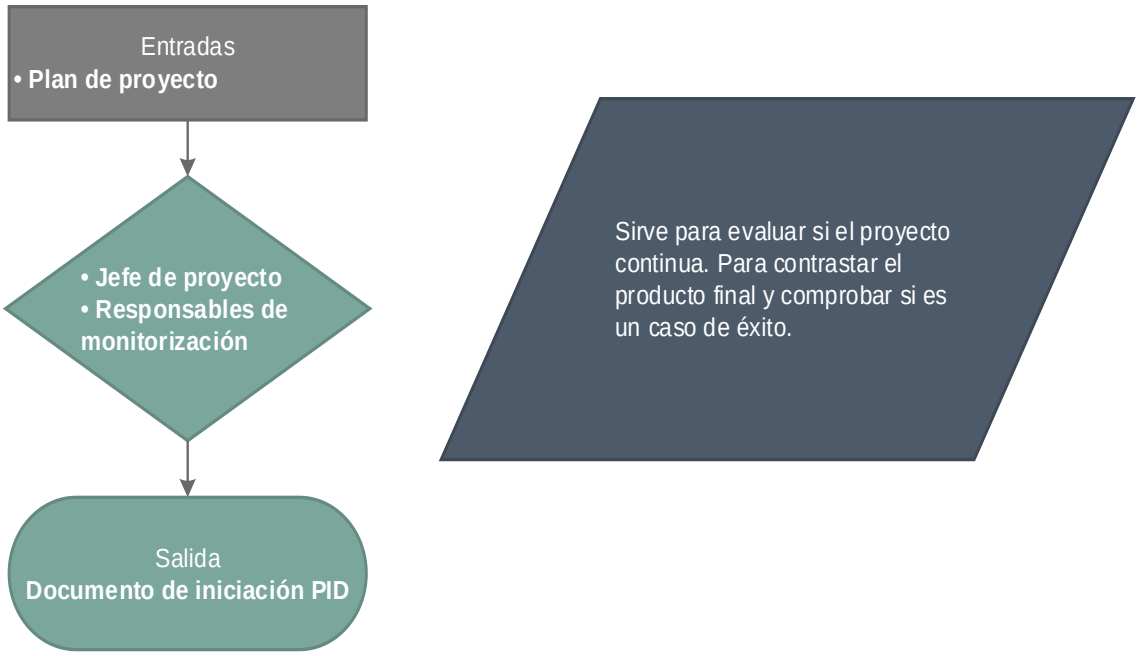


Ilustración 3.16. IP6 Ensamblar el documento de iniciación (autor)

3.2.3.7 CS1 Autorizar un paquete de trabajo

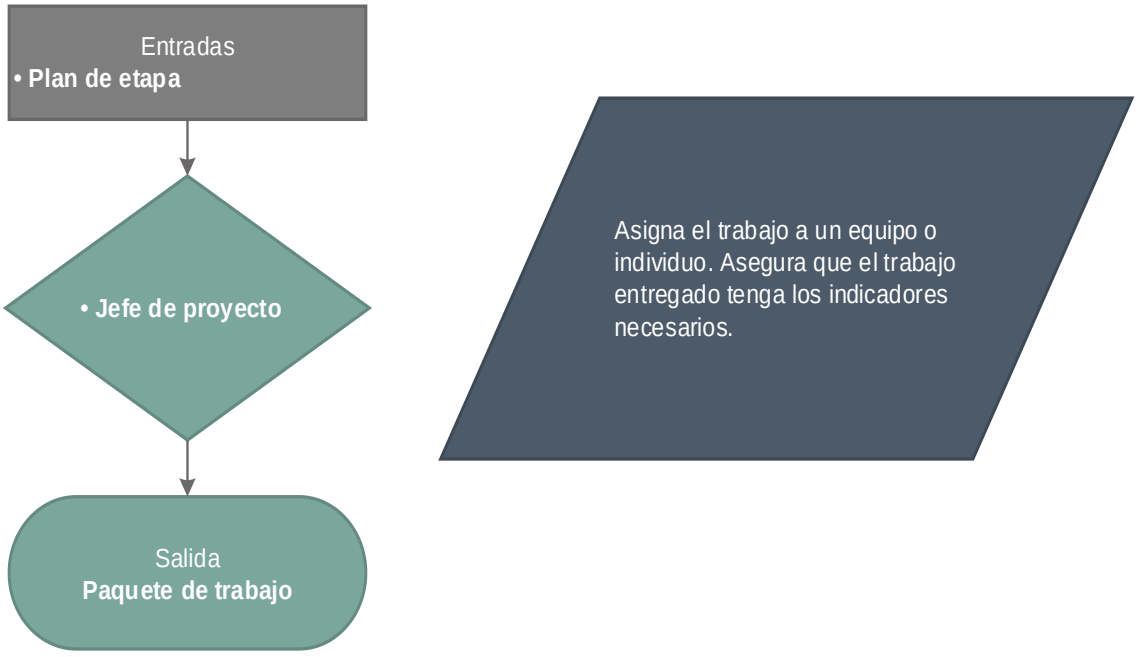


Ilustración 3.17. CS1 Autorizar un paquete de trabajo (autor)

3.2.3.8 DP3 Autorizar una etapa o plan de excepción

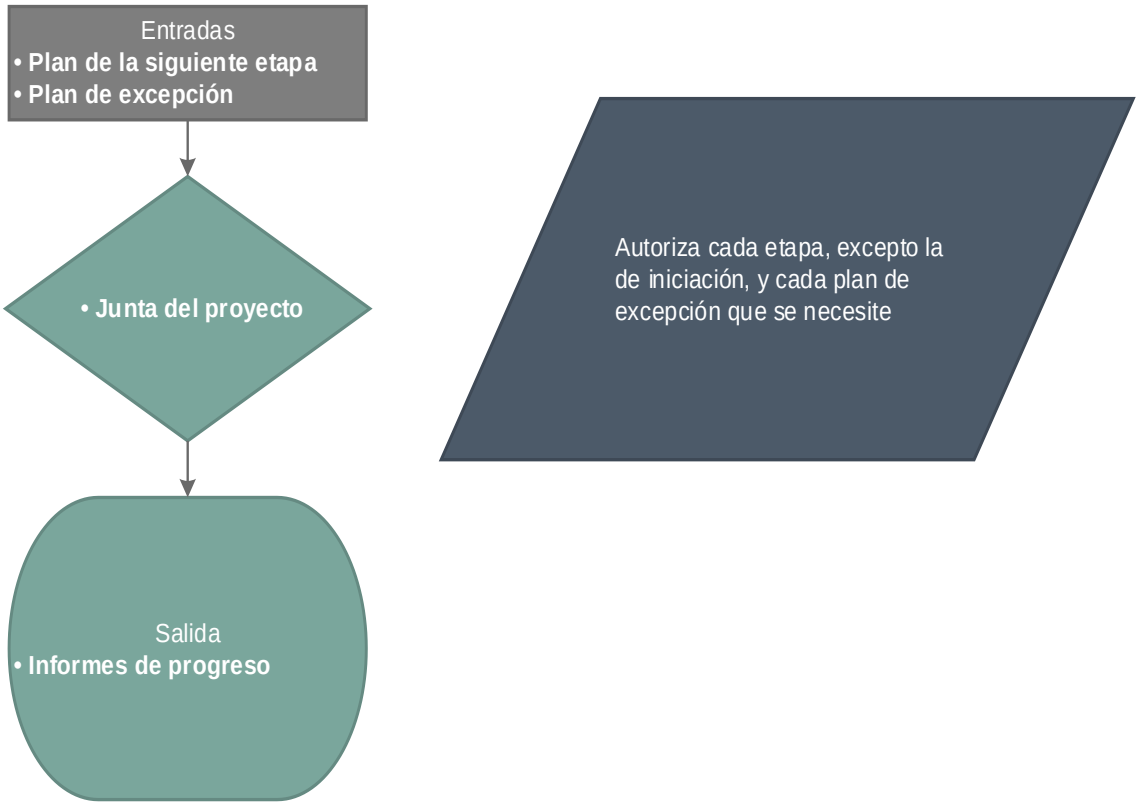


Ilustración 3.18. DP3 Autorizar etapa o plan de excepción (autor)

3.2.3.9 MP1 Aceptar un paquete de trabajo

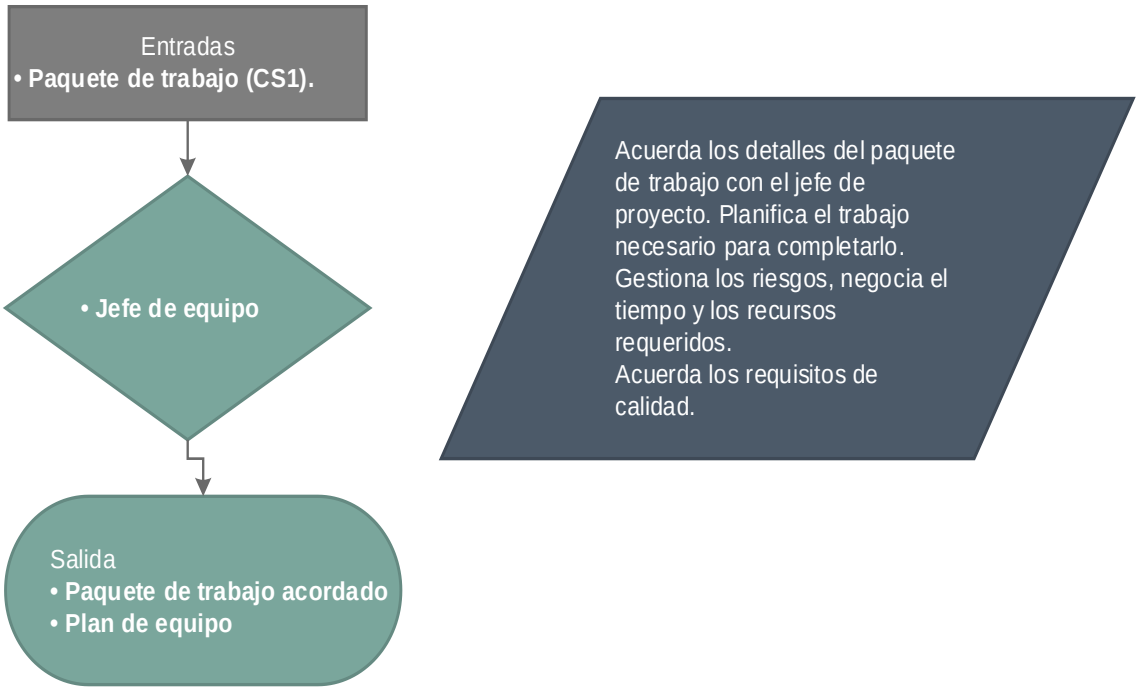


Ilustración 3.19. MP1 aceptar un paquete de trabajo (autor)

Como vemos, en Prince 2, no hay un subproceso específico para la toma de requisitos, ni una recomendación explicita. Esto se deja en manos del equipo de desarrollo y entrará dentro de su planificación de equipo. La planificación en Prince2, es a más a alto nivel.

3.2.4 Tratamiento en una metodología ágil

En Scrum, esta parte se conoce como 'pre-game' y será donde se defina el Backlog y los diferentes Sprints, pero no las tareas que compondrán dichos Sprints, se divide el proyecto en tareas, y se hace una estimación inicial de la duración del proyecto.

Se crea el Product Backlog, requisitos desde el punto de vista del cliente, lista de funcionalidades o historias de usuario que desea obtener el cliente ordenadas por la prioridad que les da el cliente. Como se analizó antes en este documento, en el apartado 3.2.3.2, se trata de la reunión de "*grooming*" entre el propietario del producto y el equipo de desarrollo. Se conforman las historias de usuario que formarán parte de la pila del producto.

3.3 **Análisis**

Es en esta etapa donde el responsable de outsourcing analizará el problema, desgranará los trabajos a desarrollar, hará una previsión de la carga de trabajo y los tiempos que llevará al equipo desarrollar la solución. Partir de una buena especificación simplificará enormemente esta fase, de lo contrario, será complicado entender de forma detallada cuál es la problemática a resolver.

A pesar de no ser complicada, contando con un analista experimentado y una buena especificación, si es crítica para el proyecto, porque un mal análisis conlleva retrasos en el proyecto, mayores costes, pero sobre todo el encarecimiento de su mantenimiento y evolución.

Una vez que existen unas buenas especificaciones, el 80% del éxito de un proyecto dependerá de un buen análisis y tener bien definidas las historias de usuario. Si no está bien analizado el proyecto, es muy probable que haya que rehacer código y automáticamente se generarán retrasos en el proyecto.

La fase de análisis deberá ser una constante a lo largo de todo el proyecto, siendo importante analizar cada vez que surja una duda, haya un cambio o se detecte una posible mejora o un problema del diseño inicial.

Al final del análisis tanto el cliente como el equipo de desarrollo, deberá tener claro cuáles son las funcionalidades que se van a desarrollar, estructura de datos, estructura de procesos, plan de pruebas, plan de integración y plan de puesta en marcha.

3.3.1 Principales problemas detectados

El principal riesgo que con lleva un mal análisis del proyecto es que no se consideré bien la complejidad del producto y esto acarreé una carga mayor del proyecto y un retraso en su realización.

Un mal análisis también puede acarrear unas malas estructuras que con lleven un mantenimiento tedioso y muy costoso. Aunque el producto funcione, a la hora de evolucionarlo o mantenerlo generará mayores costes y problemas.

Es fundamental una buena comunicación entre el cliente, el analista y el equipo de desarrollo para que todos estos implicados tengan muy claro que es lo que se va a desarrollar y de qué manera se va a hacer.

Lo habitual en la actualidad, es que una vez se entrega la especificación, ya no se tenga mucha más comunicación entre el cliente y el responsable del outsourcing, por lo que se incurre en muchos riesgos para el producto final.

3.3.2 Riesgos en esta fase

- Uno de los riesgos principales de esta fase es la mala comunicación, puede ser entre el cliente y el analista, o entre el analista y el equipo de desarrollo. Esta mala comunicación, y no tener unos procedimientos adecuados para intentar asegurarla puede dar lugar a desastres en cuanto a retrasos en el proyecto, tener que rehacer código, o presentar proyectos que poco tienen que ver con lo que se había solicitado.
- Otro riesgo habitual de esta fase es el de incluir más funcionalidad de la acordada por el cliente, en este caso *menos, es más*, dar funcionalidades que no se han pedido, puede hacer más tedioso el software, más costoso su mantenimiento, y dar lugar a software menos accesible y con mayor probabilidad de error. Para aportar valor al cliente, se deben acordar previamente esas funcionalidades con el cliente, en el análisis preliminar, pueden ser funcionalidades importantes que no fueron detectadas por el cliente, pero siempre deben estar acordadas con él en fases previas.
- También se debe ser muy riguroso en esta fase con las especificaciones del cliente, analizar si son viables técnicamente, por recursos, por coste, por entorno, etc. Si no se consideran en su justa medida estos factores, puede ser que se presupueste mal el proyecto, se incurra en retrasos en la entrega, o en el peor de los casos, que tras un período de desarrollo se llegue a la conclusión de que es un proyecto irrealizable.

3.3.3 Tratamiento en Prince 2

Prince 2 no trata esta fase de forma explícita, es el equipo de desarrollo junto con el jefe de proyecto quiénes lo abordaran en su planificación de etapa.

Al dividir el proyecto en etapas, dentro de cada etapa, existen una serie de controles y planificaciones que forman parte a su vez de este análisis.

Prince 2 aborda el control del proyecto a un nivel más alto, no entra en el detalle del análisis de las tareas, se considera que esto forma parte de los planes de equipo. Aunque si utiliza dicho análisis para valorar la salud del proyecto.

Lo que si establece Prince2 es como se dirige esta etapa que forma parte del desarrollo del proyecto (DP), desde el punto de vista de la junta del proyecto

- Proporciona vinculación con la gestión corporativa
- Aconseja al jefe de proyecto de posibles eventos del negocio o externos que puedan provocar un impacto en el proyecto
- Aprueba los planes de etapa
- Aprueba los cierres de etapa
- Decide cualquier cambio para productos ya aprobados
- Aprueba los planes de excepción, si se producen
- Da consejos y directivas a medida.
- Salvaguarda los intereses del cliente y del proveedor.
- Aprueba el cierre del proyecto.

Desde el punto de vista del jefe de proyecto (PL)

- Define los niveles de planificación necesarios para el proyecto
- Define las herramientas de planificación y los métodos de estimación que se utilizarán
- Identifica los productos cuya entrega debe ser planificada
- Identifica las actividades necesarias para entregar los productos
- Estima el esfuerzo necesario para cada actividad
- Asigna las actividades a recursos y programa la temporización de las actividades

3.3.3.1 DP4 Dirección a medida

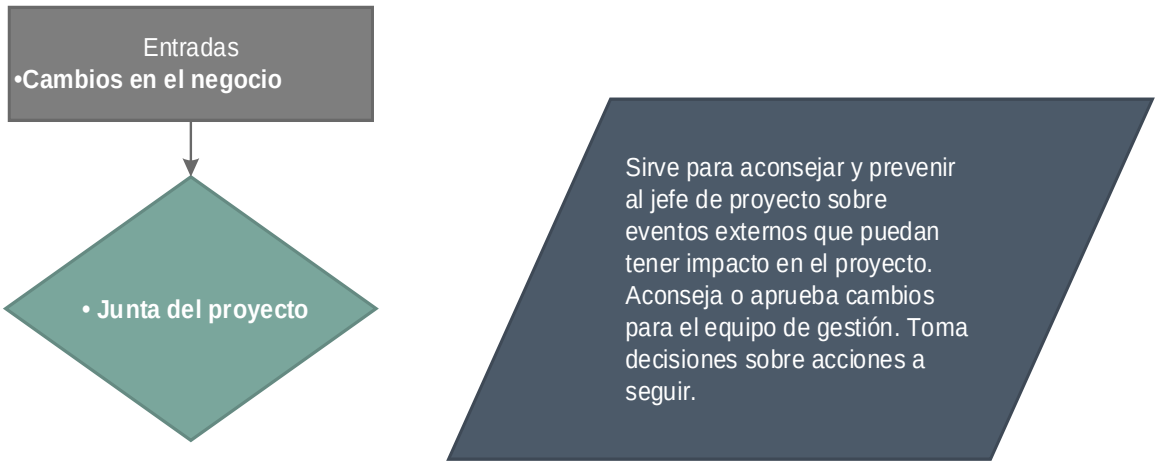


Ilustración 3.20. DP4 Dirección a medida (autor)

3.3.3.2 PL1 Diseñar un plan

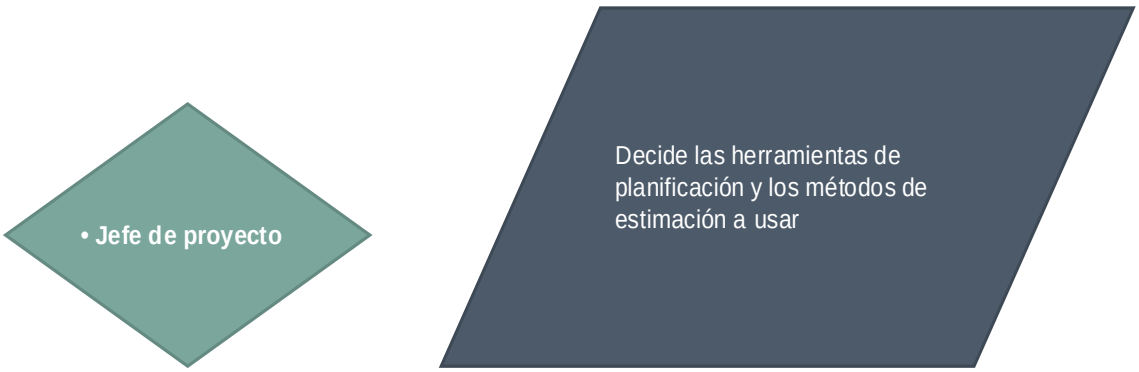


Ilustración 3.21. Subproceso PL1 (autor)

3.3.3.3 PL2 Definir y analizar los productos

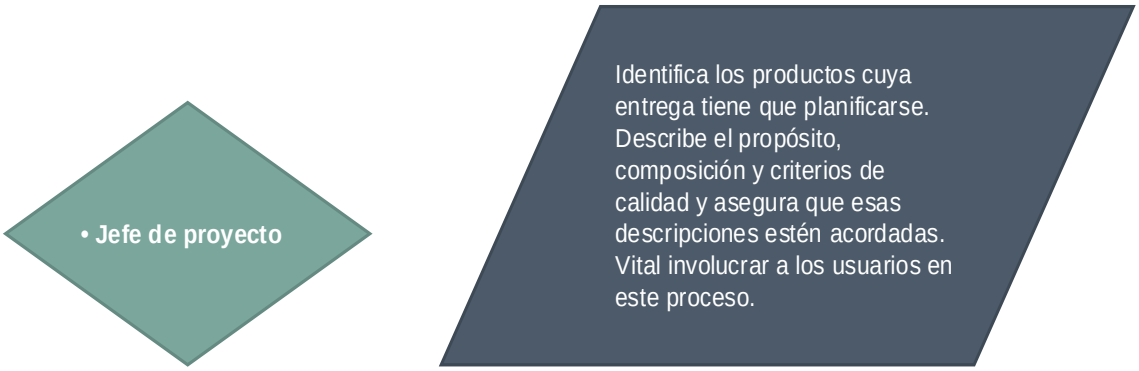


Ilustración 3.22. Subproceso PL2 (autor)

3.3.3.4 PL3 Identificar actividades y dependencias

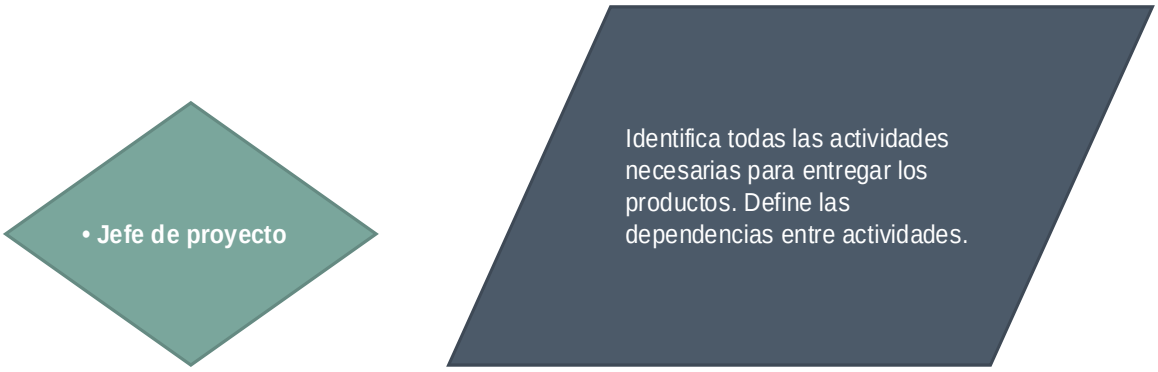


Ilustración 3.23. Subproceso PL3 (autor)

3.3.3.5 PL4 Estimar

Ilustración 3.24. Subproceso PL4 (autor)

3.3.3.6 PL5 Programar

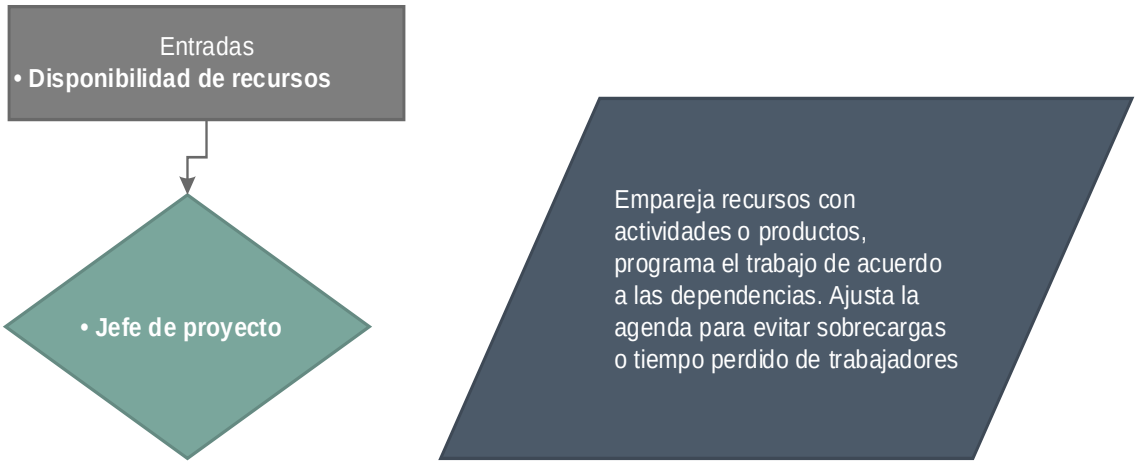


Ilustración 3.25. Subproceso PL5 (autor)

3.3.3.7 PL6 Análisis de riesgos

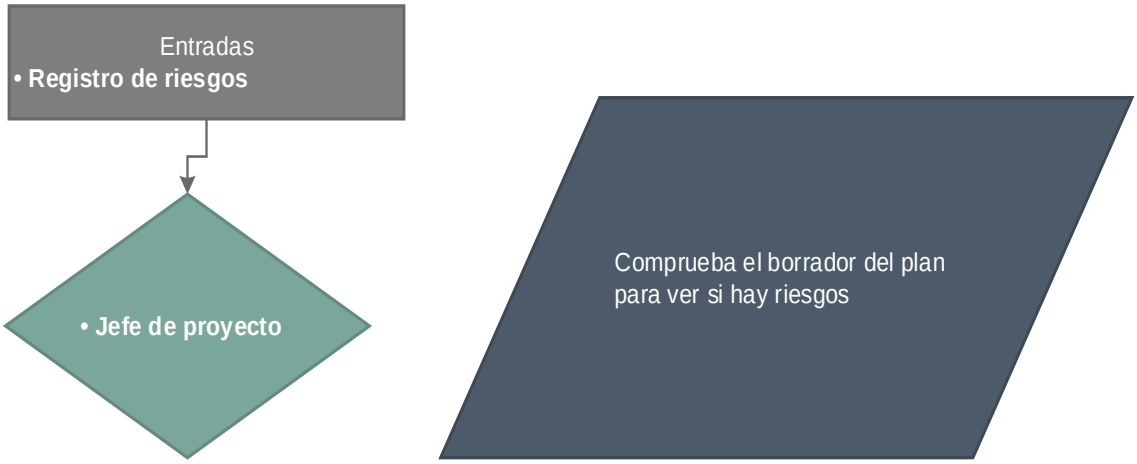


Ilustración 3.26. Subproceso PL6 (autor)

3.3.3.8 PL7 Completar el plan

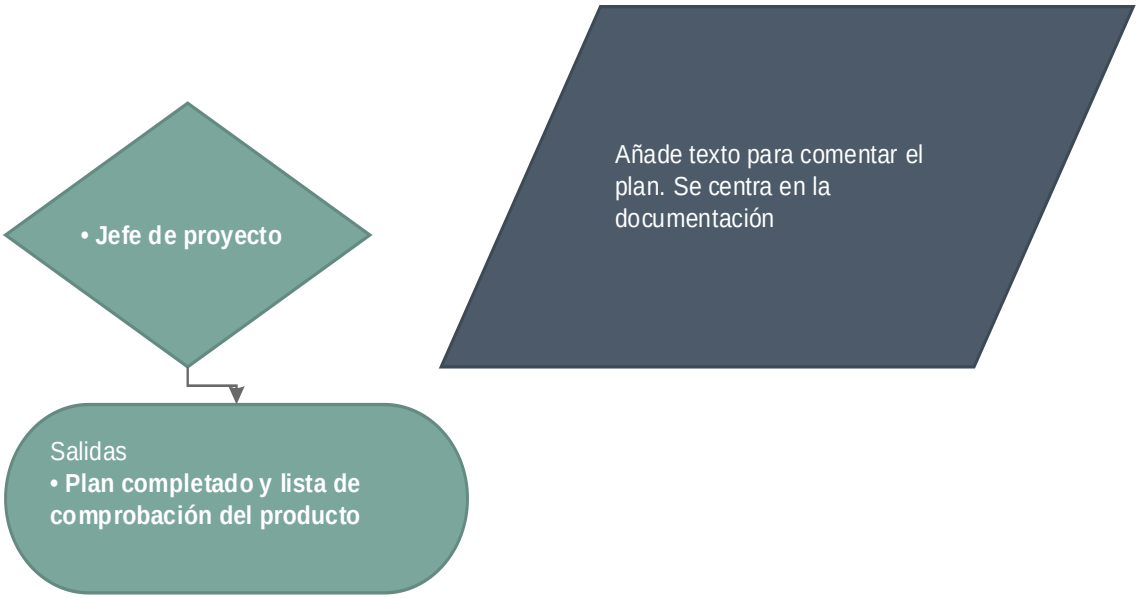


Ilustración 3.27. Subproceso PL7 (autor)

3.3.4 Tratamiento en una metodología ágil

Al trabajar en iteraciones, la etapa de análisis se aborda en las reuniones de Sprint junto al cliente como se analizó en el apartado Eventos

Al estar el cliente presente en todo momento, es fácil que detecte si no se comprendió bien alguna historia, o si no se está considerando cierta funcionalidad, y esto se resuelve dentro de la propia reunión con el objetivo de que todo el equipo de desarrollo haya entendido bien el resultado final que se pretende y a su vez, el cliente, sea consciente de las dificultades y la carga de trabajo a la que se enfrenta el equipo de desarrollo.

3.4 **Diseño software**

Una vez que los desarrolladores entienden el problema y el análisis que de él se ha hecho, deberán determinar una estrategia para resolverlo, esta etapa es el "*Cómo*" se va a solucionar.

Las actividades de diseño varían en función del proyecto, pero las habituales son (Baldonedo, 2017):

Diseño de arquitectura: identifica los principales componentes, sus relaciones y como se distribuyen.

Diseño de interfaz: se define los elementos para intercambiar información entre los diferentes componentes que forman el sistema.

Diseño de componentes: Cada módulo del sistema, se diseña su arquitectura

Diseño de datos: estructura de los datos, no tiene por qué ser en base de datos, aunque es lo más habitual.

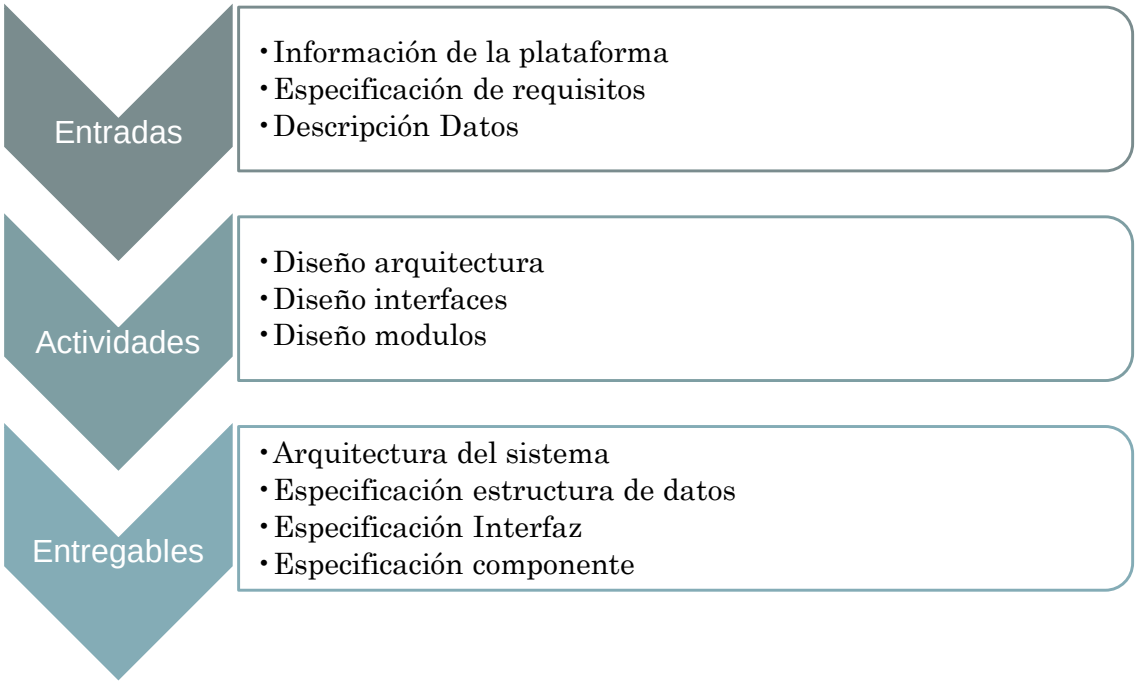


Ilustración 3.28. Diseño de software (autor)

3.4.1 Principales problemas detectados

Carencia de un diseño claro y bien documentado que sirva a los programadores para tener muy claro cómo desarrollar el producto.

Por las prisas en el desarrollo, se suele tender a hacer diseños que cumplan con la inmediatez, pero poco integrados en la arquitectura y poco modulares. Esto hace que en desarrollos posteriores se tienda a repetir el mismo código y los mismos errores, encareciendo el mantenimiento y mermando la calidad y escalabilidad del código.

Como se puede ver, según avanzamos fases, unas dependen de las otras, es decir, para poder tener un buen diseño, es necesario un buen análisis y unas buenas especificaciones. Y un mal diseño, provocara un mal desarrollo.

3.4.2 Riesgos en esta fase

- El principal riesgo es que se parta de un mal análisis.
- No obtener un buen diseño si la persona encargada no es un programador experimentado y buen conocedor de la tecnología empleada. Normalmente, además es recomendable, el diseñador del software es el propio desarrollador, por lo que no debería ser un desarrollador junior.

3.4.3 Tratamiento en Prince 2

Se puede considerar el principio de una etapa, y forma parte de la planificación del equipo.

3.4.3.1 SB1 Planificar una etapa

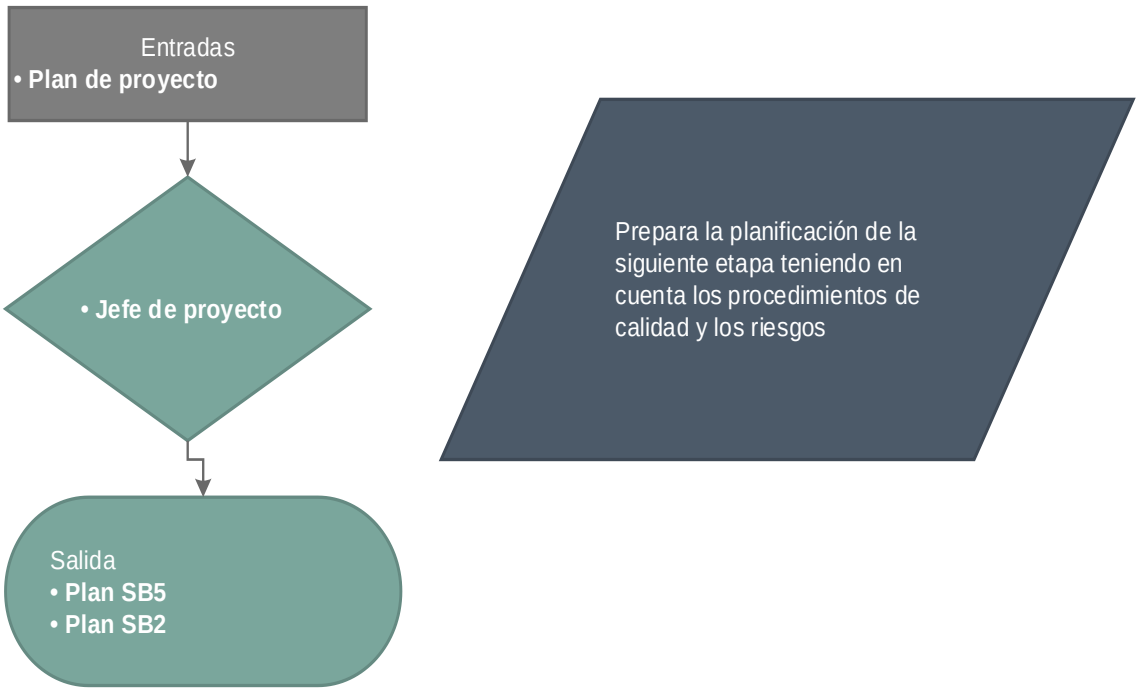


Ilustración 3.29. Subproceso SB1(autor)

3.4.4 Tratamiento en una metodología ágil

En Scrum los equipos son autoorganizados, y es tarea del desarrollador responsable de hacer la tarea, llevar a cabo también su diseño. Son tareas pequeñas, que forman parte de una historia de usuario, y está historia se desgranó y se debatió con el resto del equipo y con el cliente, durante la reunión de Sprint.

3.5 **Diseño de la interfaz**

Aunque es algo que se puede tratar como parte del desarrollo, este autor prefiere separarlo en dos fases porque esta fase en la mayoría de los casos se salta, sin embargo, es una fase importante, donde se debería hacer un diseño del producto final, una maqueta, que ayude a resolver dudas, localizar posibles problemas, y ayude a clarificar mejor cuál va a ser el resultado final.

El diseño no es cuestión de gustos, el diseño es un proceso racional, con leyes, patrones, reglas… (Conde, 2018)

Se deben diseñar las experiencias, comportamientos, diálogos. Pensar en la historia de usuario y diseñar cuál sería una buena interacción, fiable para el caso de la industria, accesible y clara.

3.5.1 Principales problemas detectados

El principal problema, es que se tiende a saltar esta fase e ir directamente a la fase de desarrollo. Además, las empresas no suelen tener personal especializado en el diseño de software, personas con conocimientos de accesibilidad, con formación en diseño de espacios que ayuden a organizar la información que se mostrará en las pantallas y en los distintos informes. Este abaratamiento procedente de prescindir de estos especialistas deriva en una falta de calidad en el producto final, además de generar diversos productos con estilos diferentes, que provocan un coste mayor de adaptación por parte del cliente al software desarrollado.

En proyectos que consisten únicamente en procesos, que no interactúan, puede parecer que carece de sentido esta etapa. Sin embargo, el hacer una pequeña maqueta del proceso,

muy sencillo, tratando el proceso como una caja negra, y presentarla al cliente, puede ayudar a clarificar aún más el resultado que espera obtener el cliente.

En algunos casos el no desarrollar una interfaz de usuario accesible y que le resulte cómoda al operador final, puede derivar en que este no la utilice y el proyecto muera o no evolucione. Este autor considera primordial que el operador final tenga una interfaz con todo lo necesario y únicamente lo necesario, distribuido de forma que al operador le resulte su interacción intuitiva y cómoda.

3.6 **Desarrollo**

Una vez terminada la fase de diseño, comienza su desarrollo. El responsable del proyecto reparte las tareas entre los miembros de su equipo, y estos comienzan a programarlo.

3.6.1 Principales problemas detectados

Es habitual que el desarrollo se haga rápido en un intento de reducir costes de programación, con mano de obra inexperta y muy cambiante. Un problema añadido es que dichos desarrolladores en la mayoría de los casos no conocen las necesidades reales del cliente ni su negocio, por lo que no son capaces de percibir los errores en el diseño o en la especificación, trasladándose estos errores al producto final.

Un ejemplo, es un informe de producción, donde se muestran toneladas consumidas inferiores a las toneladas producidas, por ejemplo. Un desarrollador desconocedor del mundo industrial, no se percata de la anomalía y por tanto no podrá alertar del problema.

3.6.2 Riesgos en esta fase

- Mala calidad del software final, poco reutilizable, en muchos casos falto de estilo de programación, poco escalable, poco seguro, nada robusto.
- Por desconocimiento del desarrollador se pueden llegar a tocar puntos vitales para el proceso y provocar errores en producción que hasta ese momento funcionaban correctamente, es decir, al no tener un buen conocimiento de la arquitectura, se pueden modificar objetos base, que tengan consecuencias en algún objeto que herede de éste y que el desarrollador no se percate, del problema que esto ocasiona. Un ejemplo real se fue un control que era común para dos instalaciones diferentes, un desarrollador de una de ellas lo modificó para que contemplase una nueva funcionalidad, lo probó y pasó todos los test en esa instalación, pero el desarrollador no era consciente que también se usaba en

otra instalación, y en esta, de manera diferente, por lo que causo una parada y las consecuencias pérdidas económicas derivadas de dicha parada.

3.6.3 Tratamiento en Prince 2

Al igual que se vio en los apartados anteriores, Prince 2, lo trata como una etapa más del desarrollo.

3.6.3.1 MP2 Ejecutar un paquete de trabajo

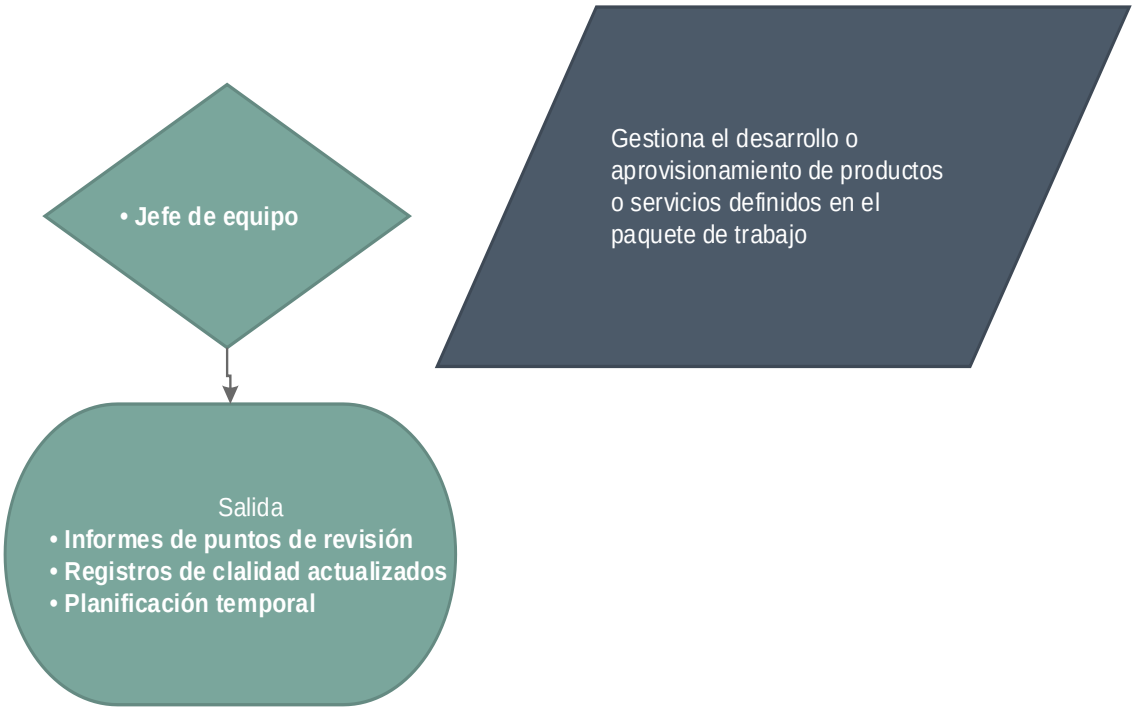


Ilustración 3.30. MP2 Ejecutar un paquete de trabajo (autor)

3.6.3.2 CS2 Evaluar el progreso

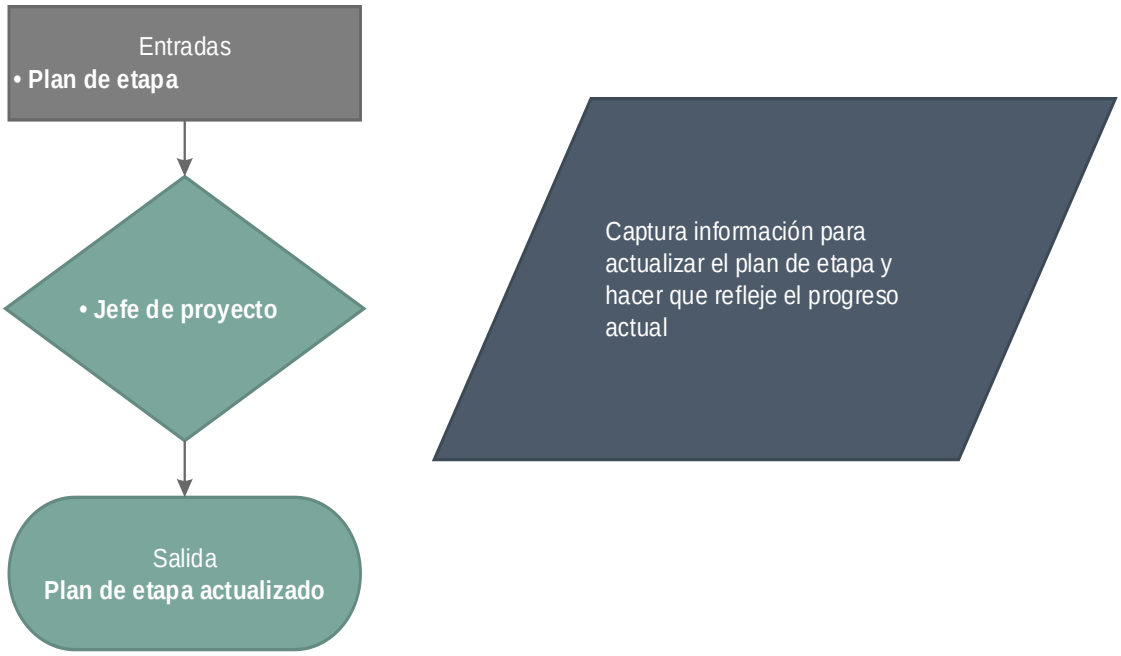


Ilustración 3.31. CS2 Evaluar el progreso (autor)

3.6.3.3 CS3 Capturar incidencias del proyecto

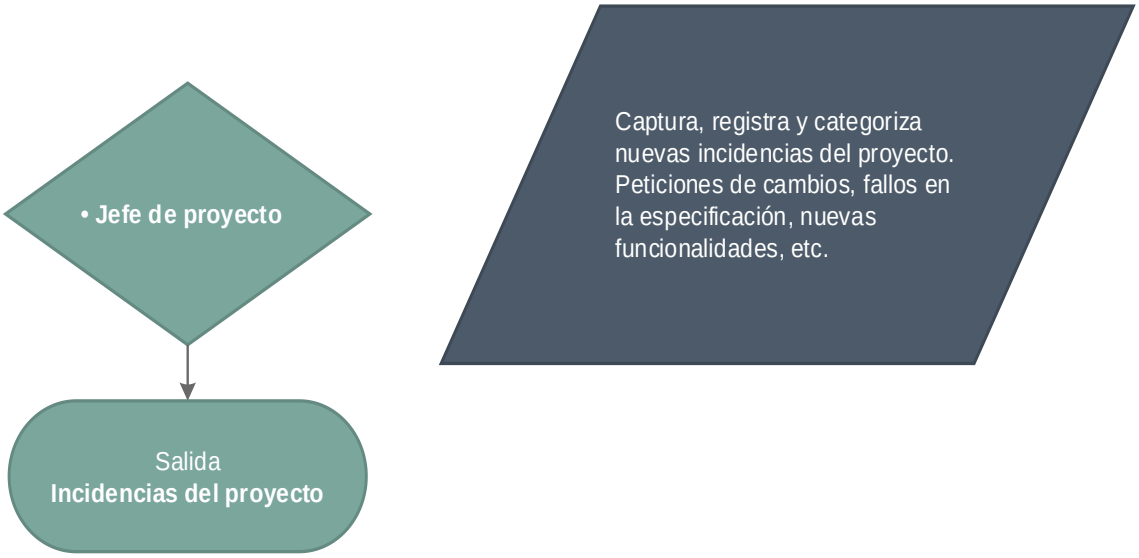


Ilustración 3.32. CS3 Capturar incidencias del proyecto (autor)

3.6.3.4 CS4 Examinar las incidencias del proyecto

Ilustración 3.33. CS4 Examinar las incidencias del proyecto (autor)

3.6.3.5 CS5 Revisar el estado de la etapa

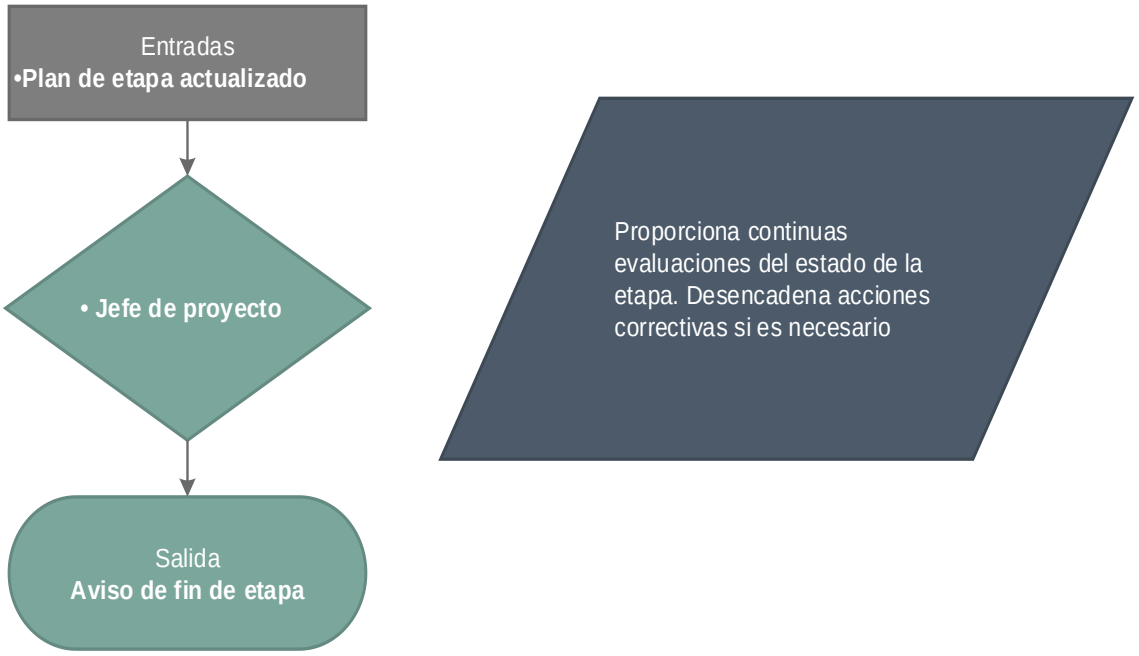


Ilustración 3.34. CS5 Revisar el estado de la etapa (autor)

3.6.3.6 CS6 Redactar informes sobre los temas más relevantes

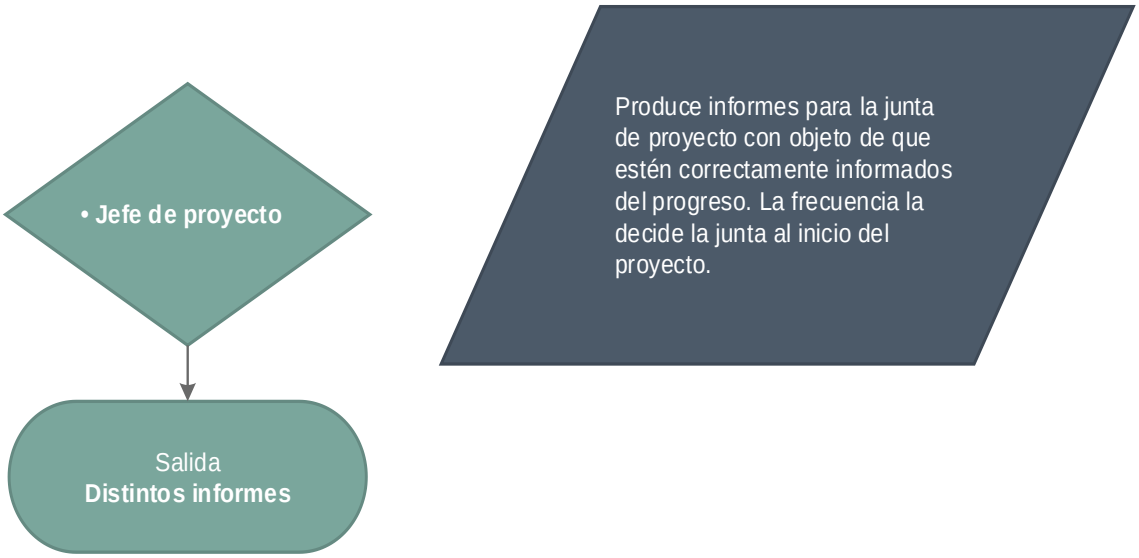


Ilustración 3.35. CS6 Informes (autor)

3.6.3.7 CS7 tomar acciones correctivas

Ilustración 3.36. CS7 acciones correctivas (autor)

3.6.3.8 CS8 Elevar incidencias del proyecto

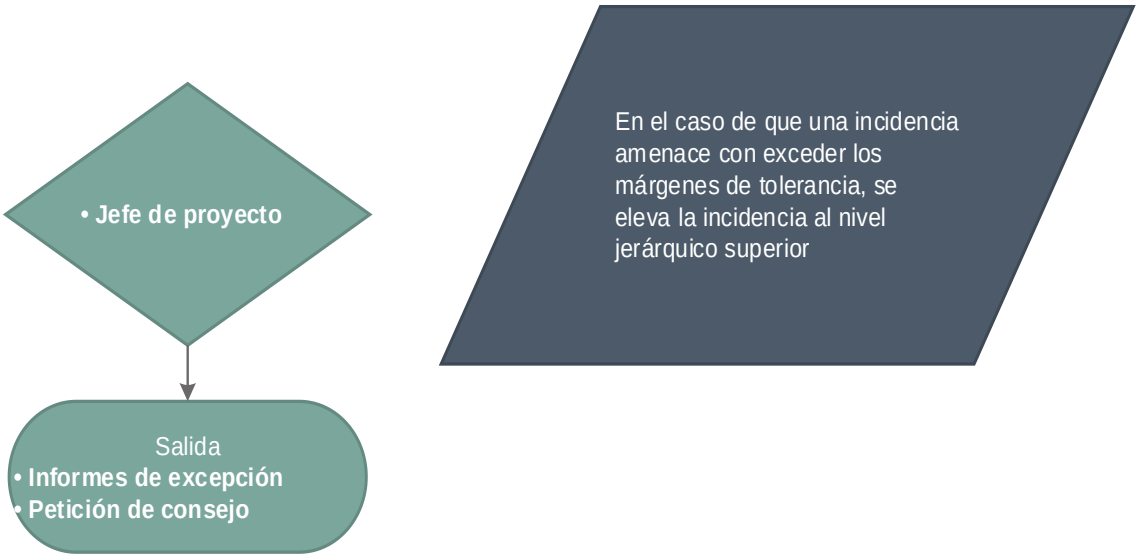


Ilustración 3.37. CS8 Elevar incidencias del proyecto (autor)

3.6.4 Tratamiento en una metodología ágil

Los equipos descritos por Nonaka y Takeuchi (Nonaka, 1986) y los principios de agilidad no prescriben el uso de una determinada táctica para lograr un desarrollo incremental. De hecho contemplan el avance constante sin incrementos.

La gestión visual Kanban es la técnica más empleada actualmente para regular un flujo de avance continuo en proyectos TIC.

El término Kanban lo empleo Taiichi Onho (Toyota) para referirse al sistema de visualización empleado en los procesos de producción, para evitar sobreproducción y almacenamiento innecesario de producto. Su origen es de finales de los cuarenta.

Kanban muestra y gestiona el flujo de avance y entrega y ayuda a evitar los cuellos de botella y tiempos muertos.

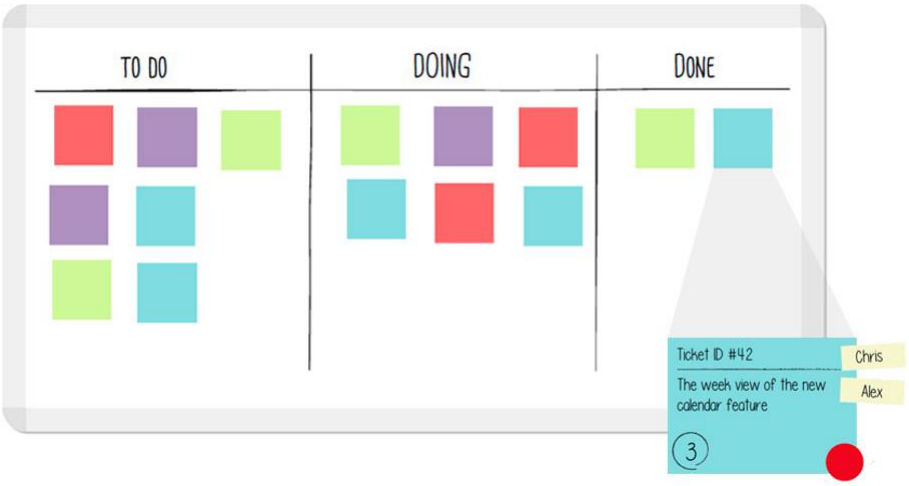


Ilustración 3.38. Tablero Kanban (Canal, 2016)

Sobre un tablero Kanban se pueden establecer pautas para regular el flujo de avance de las tareas. La posición de cada tarea en el tablero representa su estado.

Kanban saca a la superficie información sobre posibles problemas, sobre priorización de los trabajos, problemas sobre el flujo, incidencias en el desarrollo, etc.

Facilita un ritmo sostenido evitando la ley de Parkinson. La ausencia de hitos temporales evita la tendencia habitual de alargar el tiempo de trabajo hasta completar el tiempo estimado, evitando *"Que el trabajo se expanda hasta llenar el tiempo que se había previsto (Ley de Parkinson)"*.

Como radiador de información, Kanban favorece la comunicación directa al actualizar este tablero durante los Daily meetings y comparte la visibilidad de la evolución del proyecto con todos los implicados.

La actualización constante ayuda a identificar los posibles impedimentos, problemas y riesgos, que de otra manera pueden pasar desapercibidos hasta que empiezan a producir retrasos inevitables.

Favorece la cultura de colaboración y transparencia, al ver todos los miembros del equipo el estado de todo es fácil que se puedan ayudar cuando detectan un retraso de uno de los miembros.

3.7 **Debugging**

Según (Boehm, 1989), se debe distinguir entre validación y verificación.

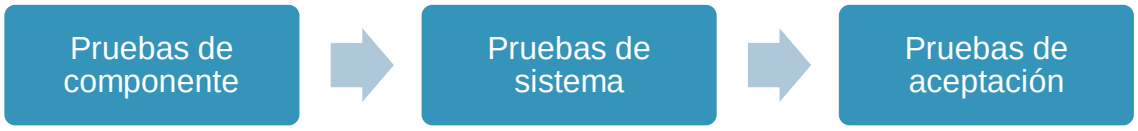
Validación → ¿Estamos desarrollando las funcionalidades especificadas?

Verificación → ¿Estamos haciendo correctamente el producto?

En esta fase, se trata de la verificación, el propio desarrollador deberá depurar bien el código intentando localizar los posibles errores que contiene el mismo. Pero en la mayoría de los casos por falta de tiempo y por ser una labor que requiere de una cierta disciplina, no se realiza con suficiente meticulosidad.

El programador tiende a pensar que su desarrollo esta carente de errores, de lo contrario, obviamente no lo hubiese desarrollado así, pero la realidad es que hay un montón de situaciones que paso por alto, y por ello es necesario una metodología que obligue al trabajador a verificar que todo está desarrollado según las especificaciones.

Las fases habituales de pruebas de software son:



3.7.1 Principales problemas detectados

- Poca dedicación de tiempo al desempeño de esta tarea tan sumamente importante y que tantos costes puede ahorrar en el futuro.

- Falta de un procedimiento o metodología de obligado cumplimiento que obligue a realizar concienzudamente esta labor

3.7.2 Riesgos en esta fase

Los riesgos de no depurar bien el código son evidentes, tarde o temprano esos errores provocaran resultados inesperados con consecuencias para la producción.

3.7.3 Tratamiento en Prince 2

Forma parte de la planificación de la etapa, y los responsables son los miembros del equipo. Los límites de calidad vienen establecidos en la petición del proyecto, así como

las tolerancias a estos límites, lo que facilita que el jefe de equipo no tenga que escalar ciertas diferencias con la calidad establecida, siempre que estén dentro de la tolerancia y forman parte del proceso de control en cada etapa Proceso de control de fase del proyecto (CS)

3.7.4 Tratamiento en Scrum

Esta fase es llevada a cabo por el miembro del equipo responsable de desarrollar la tarea. Forma parte de la etapa de desarrollo y validación de la tarea.

3.8 **Testing**

Está fase consiste en testear el nuevo desarrollo, a poder ser en un entorno similar al de producción. Esta fase es la última validación antes de la entrega del producto o de desarrollo, al cliente.

3.8.1 Principales problemas detectados

Al igual que en la fase de debugging, normalmente no se hace con la importancia que esta tarea tiene, o la hace el propio desarrollador, por lo que pasará por alto situaciones que como pasaba en la fase anterior, jamás llegó a plantearse.

No tener un entorno adecuado que simule el entorno de producción por lo que las pruebas no son reales, sino en entornos controlados con resultados esperados (Vyas, 2017).

Se realizan pruebas de todo el sistema en su conjunto, pero no pruebas unitarias, donde se pruebe por separado cada módulo que compone el sistema.

No se documentan los errores encontrados ni las causas por lo que no existe un aprendizaje real del proceso que evite los mismos errores en desarrolles posteriores.

3.8.2 Riesgos en esta fase

Un mal testeo provoca una pérdida de confianza con el cliente. Si este detecta errores sencillos frutos de un mal debugging y un mal testeo, esto automáticamente generará desconfianza con el cliente que empezará a mirar más con lupa que clase de proyectos externaliza y de qué manera lo hace.

3.8.3 Tratamiento en Prince 2

Forma parte de la planificación de la etapa, y los responsables son los miembros del equipo.

3.8.4 Tratamiento en una metodología ágil

Esta fase es llevada a cabo por el miembro del equipo responsable de desarrollar la tarea.

3.9 **Validación**

La validación consiste en entregar el software ya probado y testado al cliente. El responsable del proyecto en el cliente verificará que este software cumple con los requisitos y especificaciones solicitadas, y hará una revisión por alto para comprobar que está libre de errores.

3.9.1 Principales problemas detectados

Un problema en las fases anteriores se traduce que en esta fase se compruebe que se hayan pasado por alto requisitos o especificaciones necesarias para el proyecto y que el cliente pasó por alto o el responsable de outsourcing no interpretó bien y ahora el producto final necesita repararse de nuevo. A veces estás especificaciones son estructurales y el coste de volver a desarrollarlas o reparar el software es muy alto. Además, estas modificaciones al no estar previstas, en muchos casos se convierten en 'apaños' que estropean la estructura actual del proyecto y que con el tiempo generan problemas en el mantenimiento al no estar homogéneas al resto de código.

3.9.2 Riesgos en esta fase

El principal riesgo de esta fase es que el producto entregado al cliente no cumpla con las funcionalidades que espera, y haya que rehacerlo, lo que provoca retrasos y código desestructurado que puede incurrir en errores. Además, fruto de ese retraso o esas prisas por solucionarlo, puede que el desarrollo no mime tanto la calidad del producto dando lugar a pequeños apaños que ya convivirán siempre en ese proyecto.

Otro riesgo, es que el responsable en el cliente, que no suele ser el usuario final, tampoco se dé cuenta de que el producto no cumple con las necesidades finales, por lo que se valide

en esta fase y al usuario final le llegue un producto que no cumple con lo que necesita y que ya va a tener que convivir con él.

3.9.3 Tratamiento en Prince 2

Se corresponde con la fase de cierre de etapa. Entrega en juego el proceso para gestionar los límites de etapa (SB). O bien, con la etapa de cierre de proyecto (CP).

En caso de ser de etapa:

- Confirma a la junta de proyecto, cuántos productos se han entregado de los planificados en la etapa actual
- Proporciona las razones para no haber entregado productos planificados si los hubo.
- Verifica que cualquier lección útil se archivó en el registro de lecciones aprendidas
- Proporciona información para que la junta del proyecto verifique la viabilidad.
- Obtiene aprobación para el plan de la siguiente etapa
- Establece los márgenes de tolerancia para la siguiente etapa.

Si se trata de cierre de proyecto:

Verifica que se han entregado todos los productos y que estos han sido aceptados

Verifica que se han tratado todas las incidencias del proyecto

Registra cualquier recomendación para trabajos posteriores

Recomienda el cierre del proyecto a la junta

Planifica la medición de la consecución de los objetivos.

3.9.3.1 MP3 Entregar un paquete de trabajo

Ilustración 3.39. MP3 Entregar un paquete de trabajo (autor)

3.9.3.2 SB2 Actualizar el plan de proyecto

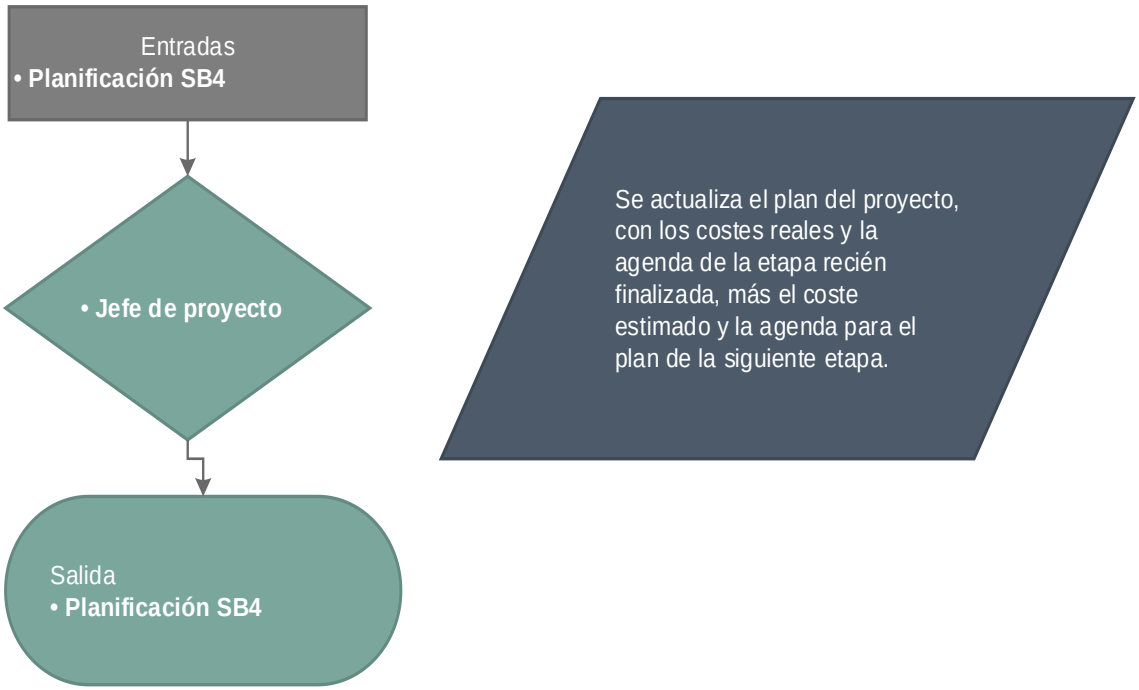


Ilustración 3.40. Subproceso SB2 (autor)

3.9.3.3 SB3 Actualizar el caso de negocio del proyecto

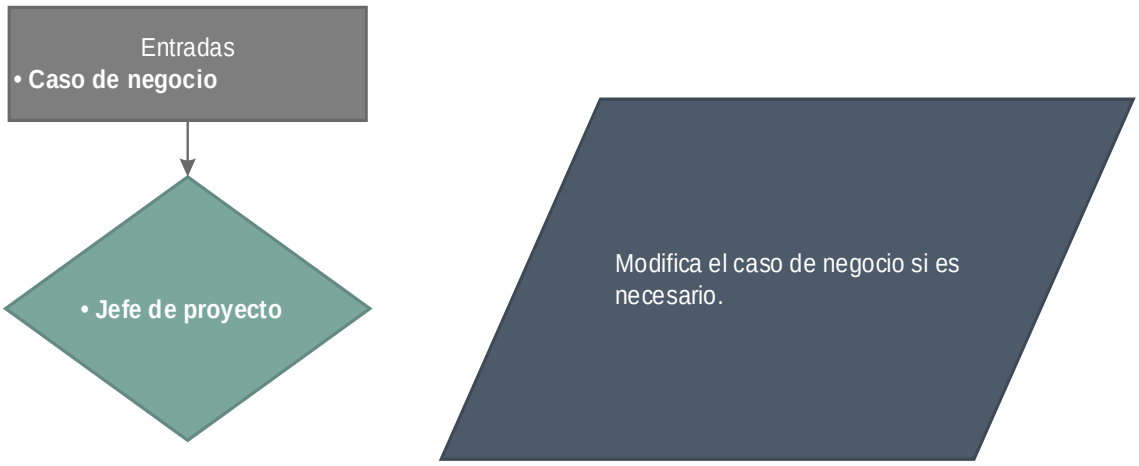


Ilustración 3.41. Subproceso SB3 (autor)

3.9.3.4 SB4 Actualizar el registro de riesgos

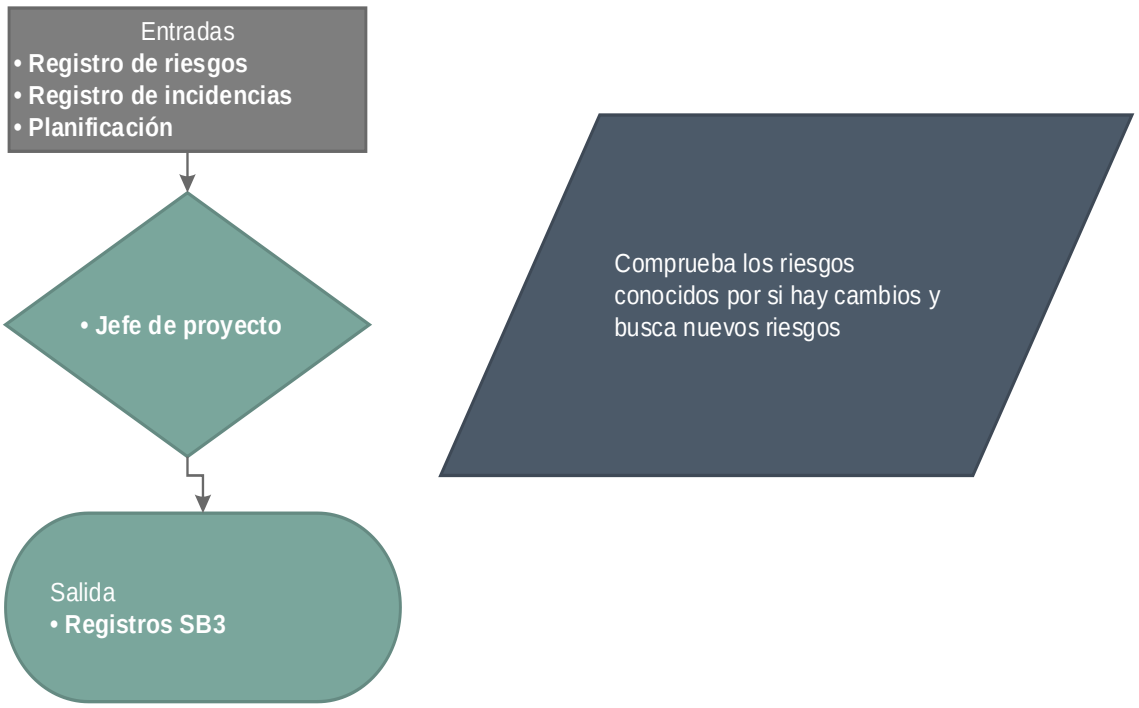


Ilustración 3.42. Subproceso SB4 (autor)

3.9.3.5 SB5 Informar del fin de etapa

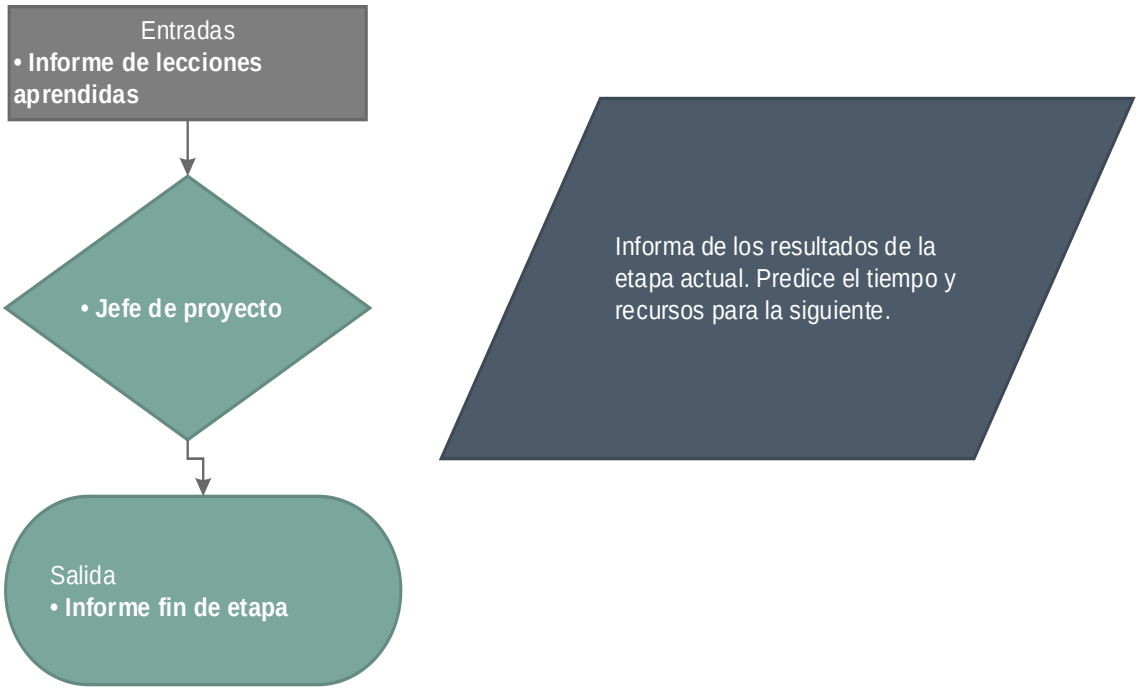


Ilustración 3.43. Subproceso SB5 (autor)

3.9.3.6 SB6 Producir un plan de excepción

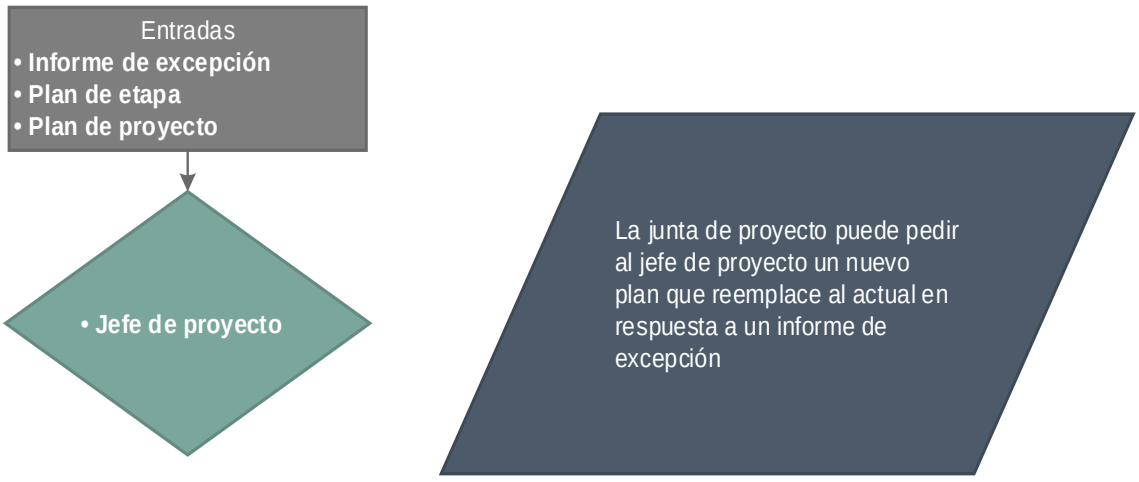


Ilustración 3.44. Subproceso SB6 (autor)

En el caso de cierre de proyecto:

3.9.3.7 CP1 Poner fin al encargo del proyecto

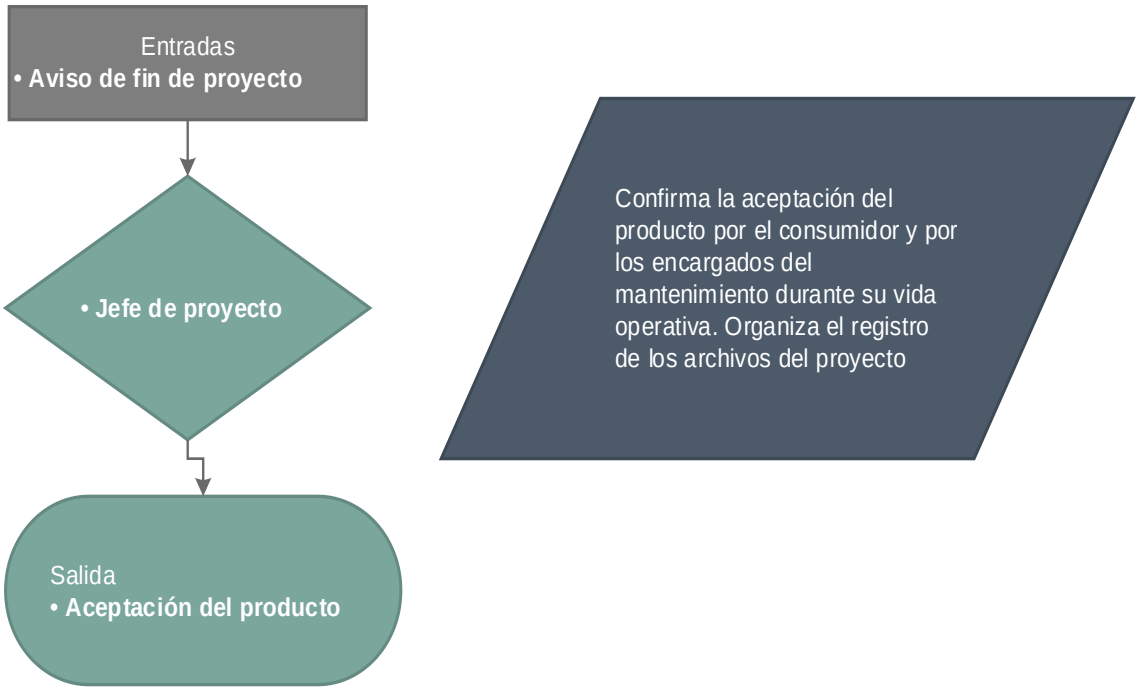


Ilustración 3.45. Subproceso CP1 (autor)

3.9.3.8 CP2 Identificar acciones posteriores

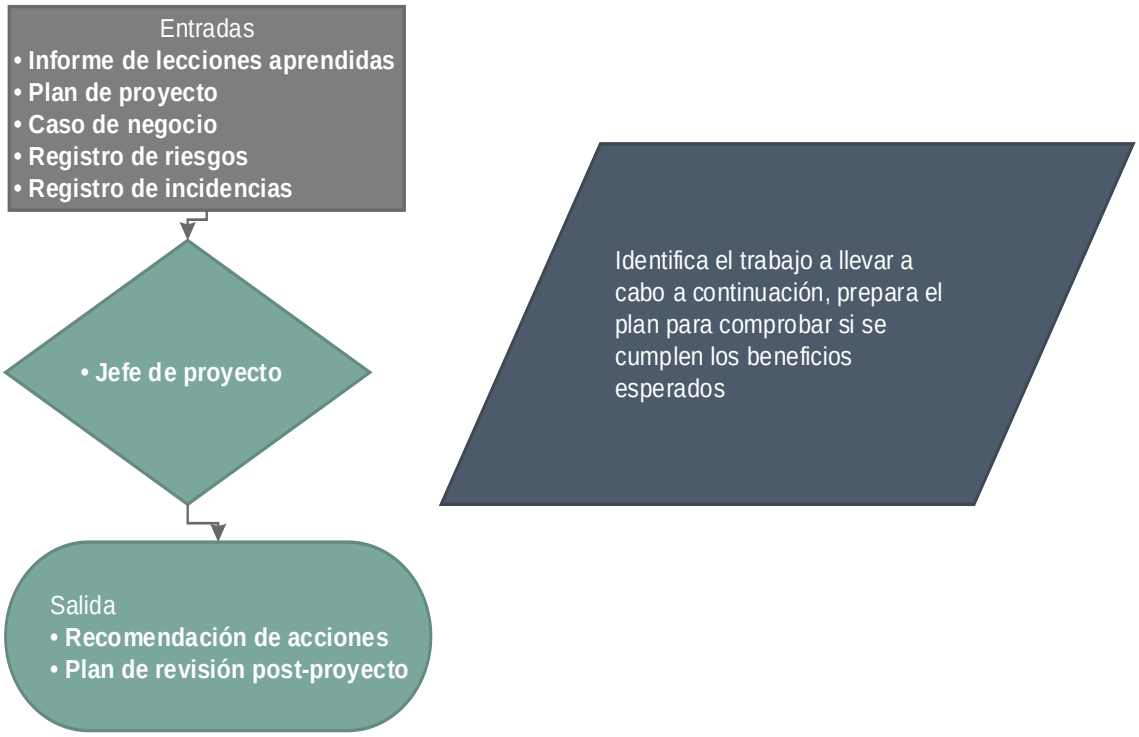


Ilustración 3.46. Subproceso CP2 (autor)

3.9.3.9 CP3 Revisión de evaluación del proyecto

Ilustración 3.47. Subproceso CP3 (autor)

3.9.4 Tratamiento en Scrum

Se corresponde con el fin de Sprint. Se entregan los productos terminados y preparados para poner en producción. Y tras esto tendría lugar la reunión de pre-sprint donde, como se vio en el apartado 3.2.3.3, se divide en dos; una primera reunión de revisión de sprint donde el propietario del producto comprueba el progreso del sistema, se identifican las historias que ya se puedan considerar hechas y las que no, y se revisa la pila del producto.

La segunda parte es la de retrospectiva, done el equipo hace un autoanálisis de su forma de trabajar, con el objetivo de corregir posibles malas prácticas y mejorar su productividad y rendimiento. Se debe centrar en "CÓMO" se está construyendo.

3.10 **Evolución**

Una vez que un desarrollo tiene éxito, lleva instalado varios meses, se usa y se obtienen resultados, es probable que se quieran añadir o mejorar nuevas funcionalidades, está es la fase de evolución.

Ilustración 3.48. Dilbert en español. Evolución producto (Adams, 2019)

3.10.1 Principales problemas detectados

Desde la empresa de outsourcing al no tener contacto con el cliente final, no se recoge feedback de este, por lo que es complicado aprender de los errores cometidos en productos anteriores y mejorar los productos futuros, o incluso hacer propuestas de mejora.

Por ejemplo, una pantalla de seguimiento de un horno, donde se han puesto colores que simulan el color del acero en función de su temperatura de calentamiento, y los desarrolladores, sienten que es un buen producto y que es una pantalla atractiva. Sin embargo, el usuario final, que tiene que pasarse casi ocho horas mirando esa pantalla, cuyos colores no le aportan nada, y sin embargo le resultan muy molestos a la vista. Pero a este usuario, nunca nadie le llega a hacer las preguntas adecuadas, que descubran que en realidad esos colores le molestan más que le ayudan, y tendrá que sufrirlos en esa pantalla y por no haber dado feedback, seguramente en pantallas venideras.

3.10.2 Riesgos en esta fase

3.10.3 Tratamiento en Prince 2

3.10.3.1 CS9 Recibir un paquete de trabajo terminado

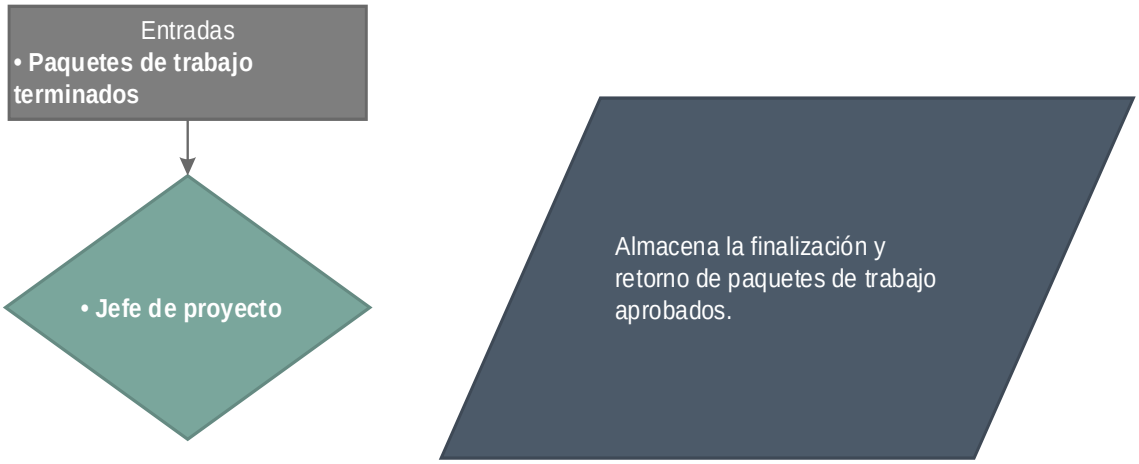


Ilustración 3.49. CS9 Recibir un paquete de trabajo (autor)

3.10.3.2 DP5 Confirmar el cierre del proyecto

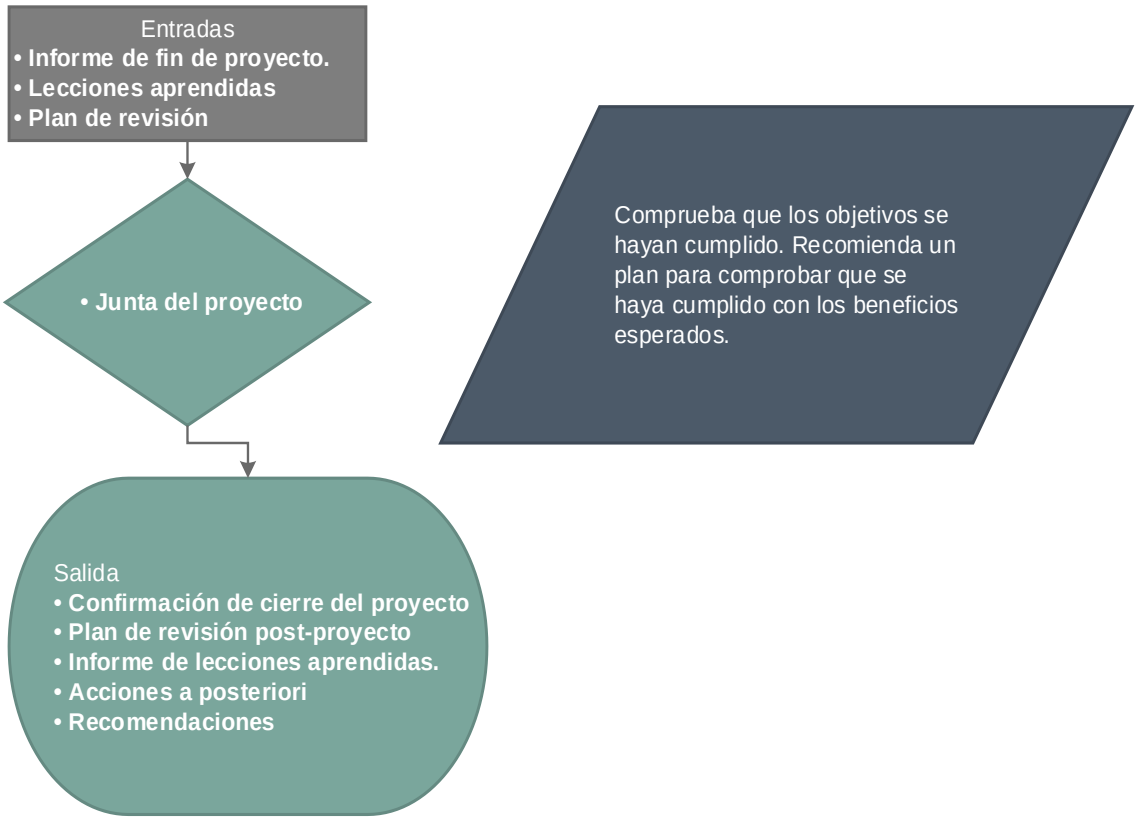


Ilustración 3.50. DP5 Confirmar el cierre del proyecto (autor)

3.10.4 Tratamiento en una metodología ágil

En Scrum, esta fase se trata como en las anteriores, si el dueño del producto decide hacer una modificación o una evolución al proyecto, lo priorizará en el 'Product Backlog' y entrará en otro Sprint donde el equipo la desarrollará.

3.11 **Resumen de la situación**

Se pudo comprobar analizando los problemas descritos anteriormente, que al no seguir una metodología de trabajo, se produce una merma de calidad, lo que repercute a su vez en un mayor coste a largo plazo. Como además no se definen con claridad los productos, ni se trata directamente con el usuario final, se alargan los tiempos hasta que el producto, que en la mayoría de los casos no es el esperado, se pone en funcionamiento.

Pero además no es suficiente trabajar con una metodología ágil o una metodología tradicional para cubrir todos los casos, sigue habiendo muchos espacios que no se cubren, que no están del todo definidos o que en la práctica sería complicado de llegar a aplicar sin aumentar de forma considerable el coste para la empresa de outsourcing.

En este gráfico podemos comprobar que procesos en Prince2 y Scrum, afectan a cada fase del ciclo de software, y quienes serían los responsables según lo visto hasta ahora.

Se comprueba que Prince2 se ocupa de gestionar el control y la estabilidad a un nivel general del proyecto, mientras que Scrum se centra en el control y agilidad del proyecto a nivel de detalle del proyecto.

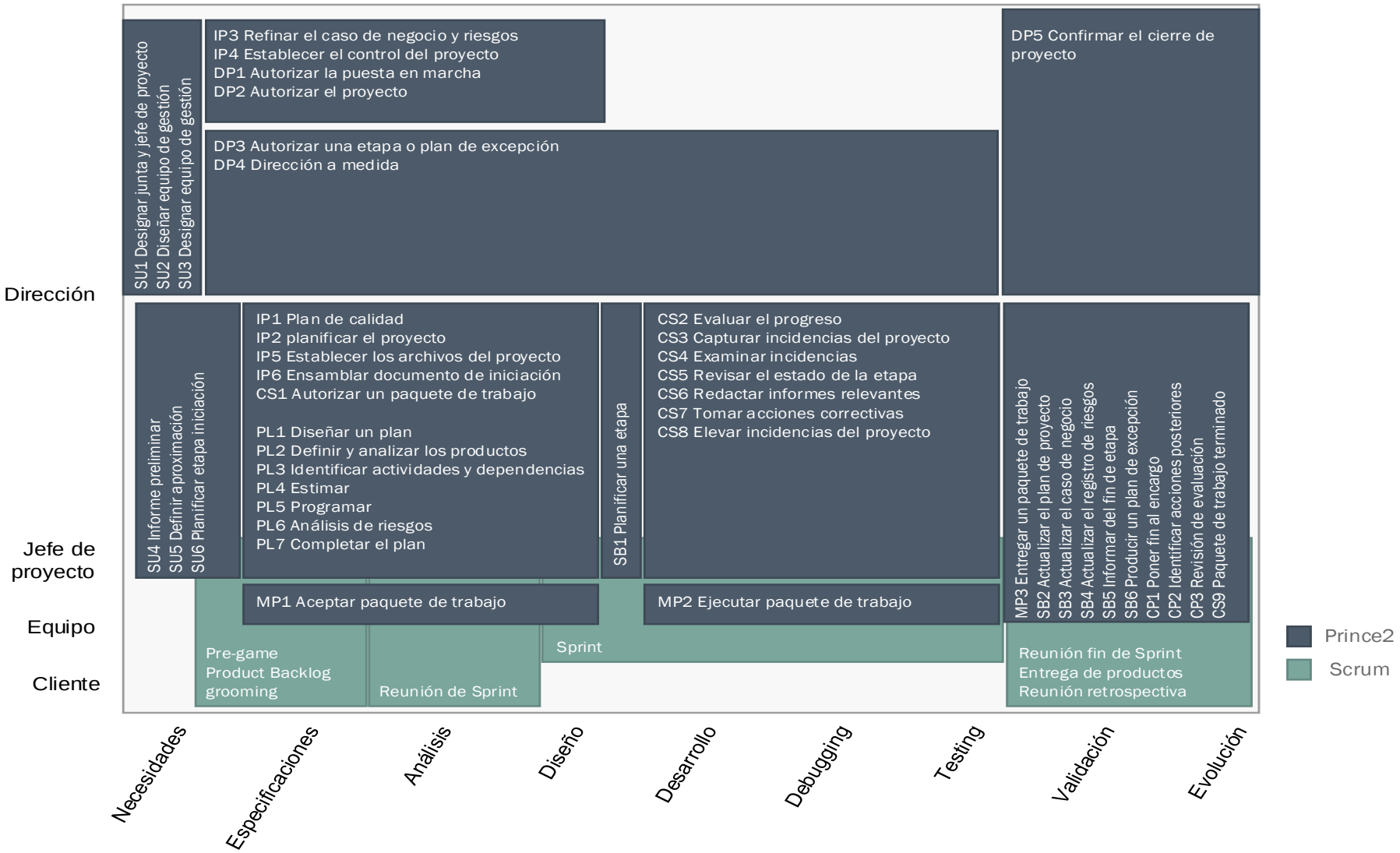


Ilustración 3.51. Gráfico de procesos asociados al ciclo del software(autor)

Si se combina Prince2 con Scrum, se puede aprovechar de Prince2:

- El estudio continuo de la viabilidad del negocio
- La integración de los stakeholders por parte de la junta del proyecto y por tanto responsabilidad compartida
- Las etapas de gestión
- Una documentación estructurada, con documentación de iniciación, paquetes de trabajo, gestión de calidad, monitorización y seguimiento.

Por su parte Scrum:

- Se centra en las necesidades del cliente
- Gestiona que se produzcan entregas a tiempo
- Equipo autogestionado
- Colaboración entre los integrantes del equipo y el propio cliente
- La calidad siempre es prioritaria
- Se desarrolla iterativamente, con tiempos establecidos
- Fomenta la comunicación continua y clara

Para integrar ambas metodologías, se deben realizar algunos cambios de concepto en ambas, y además dependerá del cliente y del proyecto, pero las principales modificaciones serían:

- En Prince2 se controla estrictamente el cambio, esto es algo que se debe modificar ya que en las metodologías ágiles se fomenta el cambio ya que se considera algo inevitable
- En Prince2 se predice desde el inicio, el alcance del proyecto, en Scrum se trabaja conjuntamente con el cliente para alcanzar el máximo valor posible.
- El riesgo se debe asumir y valorar su coste, pero no contenerlo como pasa en la filosofía Prince2
- Las lecciones aprendidas no se dejan para el final, se utilizan en cada interacción y se analizan en el final de Sprint.

- Se debe centrar en la calidad en vez de en el coste o en el tiempo.
- Se específica a un nivel alto de detalle, que luego en cada iteración se irá matizando.

4 Metodología para desarrollo de software industrial desde una empresa de outsourcing

A partir de los datos estudiados en los apartados anteriores, se propone una metodología hibrida, fundamentada en Prince2 y Scrum más Kanban, para la gestión de proyectos de corta duración.

En el siguiente apartado, se validará con un caso real, aplicando esta metodología descrita a continuación desde el punto de vista del jefe de proyecto en la empresa de outsourcing

4.1 **Objetivos**

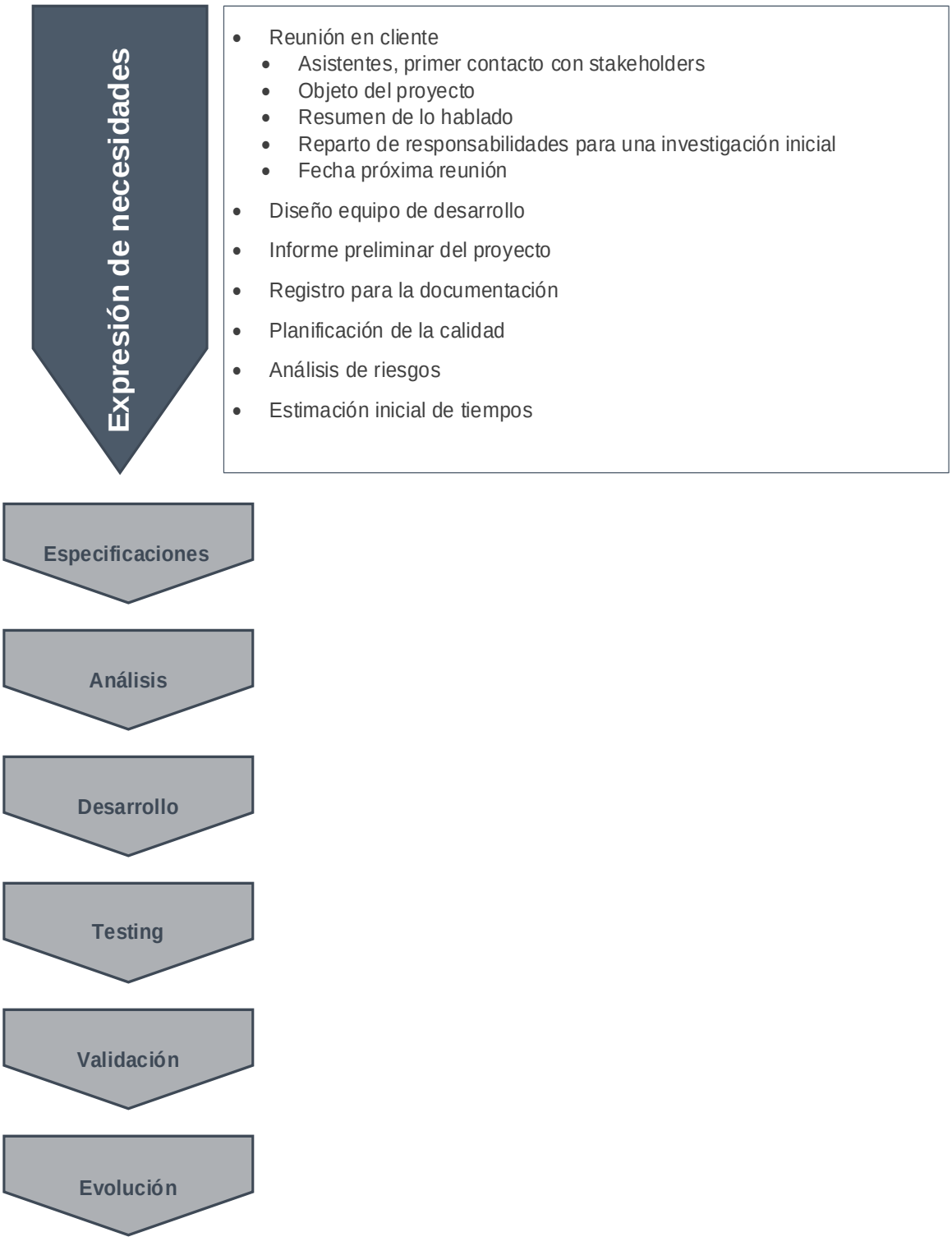
Definir una metodología y unos procedimientos basados en Prince2 y Scrum, que cubran las necesidades actuales de las empresas de desarrollo de software para la industria 4.0. De manera que se produzcan desarrollos fiables y de calidad, en un tiempo prudencial y a bajo coste.

Para ello se sigue usando el ciclo de vida del software como etapas claves a considerar, y se verá en cada una de ellas, que procesos usar y que procedimientos son los adecuados, siempre desde el punto de vista de la empresa de outsourcing.

Se intenta reducir la distancia entre el operador final y el desarrollador, minimizando la problemática existente de la falta de conocimiento del desarrollador sobre el negocio del cliente e intentando minimizar el riesgo de realizar este tipo de desarrollos cruciales para la industria, que necesariamente necesitan ser de alta fiabilidad, pero a bajo coste y con mano de obra inexperta.

Se usan métodos ágiles, y se intenta reducir burocracia y procedimientos que no aportan valor, para así reducir tiempos en los desarrollos y abaratar en lo medida de lo posible el coste de desarrollo.

4.2 **Expresión de necesidades**



Una vez que el cliente decide contratar un servicio o el desarrollo de un software, el jefe de proyecto (en la empresa externa), se reúne con el cliente y deberá recoger la necesidad del cliente.

4.2.1 Reunión en cliente

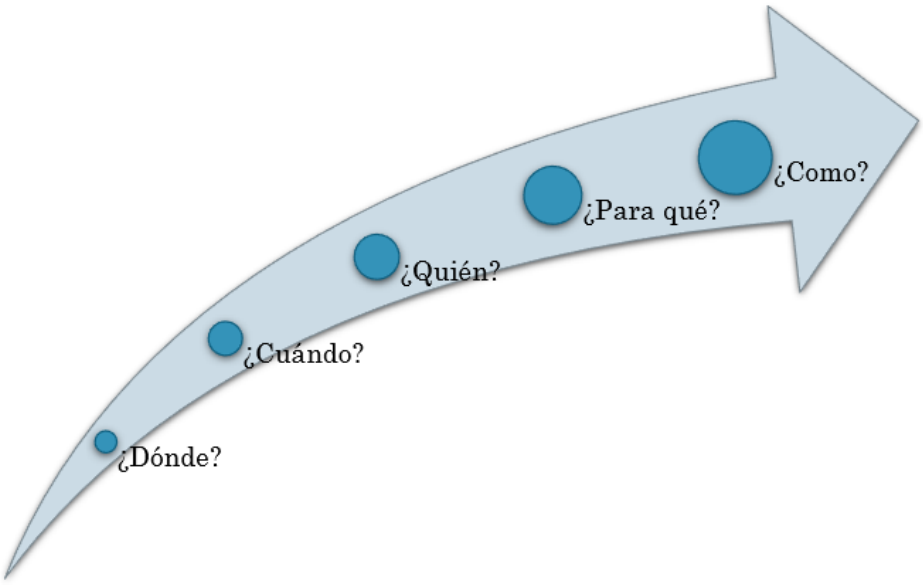


Ilustración 4.1. Esquema reunión cliente (autor)

Es importante que a esta reunión asistan los implicados por parte del cliente, es decir, será interesante que al menos acuda un usuario final del requerimiento, y por parte de la empresa auxiliar, además del jefe de proyecto, en algunos casos un especialista.

Se trata de recabar información, por lo que se delimitarán a escuchar, y hacer preguntas en aquellas cosas que no están claras. Siguiendo la regla del 80/20, escuchar el 80% del tiempo y hablar sólo un 20%.

Para terminar la reunión, se deberá hacer un resumen de lo acordado, por parte del jefe de proyecto, de forma que se demuestre al cliente que se entendió su necesidad, y que se entiende lo que el cliente espera de la empresa de outsourcing, validando así, esta primera comunicación.

Más tarde se rellenará un acta de la reunión que se enviará a todos los asistentes para que estos puedan validar que toda la necesidad ha quedado suficientemente aclarada y que no se escapa alguna cosa importante o que no se mal interpreto algún aspecto.

Esta acta deberá recoger al menos:

4.2.1.1 Fecha y lugar de la reunión

4.2.1.2 Asistentes

Es importante recoger el cargo que ostenta el asistente para ir identificando los stakeholders del proyecto, así como su contacto, el email y si puede ser también un teléfono de contacto, para agilizar lo más posible las comunicaciones con él.

En el Anexo 2 se adjunta una plantilla como ejemplo para almacenar los contactos del proyecto.

4.2.1.3 Objeto de la reunión

El objeto de la reunión será un titular a modo de resumen, que indique de que va a tratar el documento.

4.2.1.4 Resumen acordado

En esta parte se describirá la necesidad del cliente, y se explicará cuál es la necesidad del cliente, y que es lo que necesita de la empresa de outsourcing.

Es un parte importante, en muchos casos contractual, por lo que debe describir de forma muy objetiva y con el mayor grado de detalle posible, sin valoraciones ni propuestas por parte de la empresa externa, únicamente todas las valoraciones del cliente.

4.2.1.5 Reparto de tareas y responsabilidades

Es posible que durante la reunión surjan dudas, o salgan a luz puntos débiles en el proyecto, o que no están lo suficientemente aclarados. Se asignarán tareas a los asistentes, y se fijará una fecha para finalizar dichas tareas, que consistirán en búsqueda de información, o investigación sobre estos puntos que todavía no están suficientemente aclarados.

4.2.1.6 Fecha de próxima reunión.

Es importante dejar fijada la fecha de la siguiente reunión donde, si los datos recabados por la empresa de outsourcing son suficientes, esta presentará una propuesta al cliente. O bien, si quedan lagunas, se fijará una segunda reunión donde aclarar estos puntos.

En el Anexo 1. Se adjunta una plantilla que puede servir para cubrir este primer punto.

4.2.2 Diseño del equipo de gestión del proyecto

Se decide que recursos se asignarán al proyecto y se forma el equipo. No es conveniente formar equipos de más de 6 personas, en el caso de que el proyecto requiera más recursos, se recomienda subdividir el proyecto de manera que cada subproyecto este gestionado por equipos de entre 4-6 personas (How to Build a Great Team, 2006).

En 1.861 Maximilian Ringelmann (efecto Ringelmann) descubrió que cuántas más personas tiraban de la cuerda, menor era el esfuerzo que realizaba cada una de ellas, para ello analizó a diversos individuos solos y en grupo y cuantifico la fuerza ejercida en cada caso, cuando se iban añadiendo personas, la fuerza total aumentaba pero la media ejercida por cada miembro disminuía.

En 2.006 (Klein, 2006), escribió que cuando un equipo pasaba de ocho o nueve personas, es demasiado grande y se desmorona en grupos más pequeños, cuando aumenta el tamaño del equipo siempre aparece algún miembro que no cumple y se esconde entre los demás.

Según (Wittenberg, 2006), el número 6 es según sus estudios, el tamaño óptimo, ya que es el tamaño que fomenta que el equipo se conozca en profundidad, esté motivado, y tenga sensación de pertenencia a grupo. Además, valora que se tenga comunicación fuera del ámbito electrónico, y con un número superior a 6 es complicado este tipo de reuniones.

En el ejercito la menor unidad es la escuadra, formada por un cabo y tres o cuatro soldados dependiendo del arma.

Según la experiencia del autor, coincide en que los equipos para este tipo de proyectos no deberían superar las seis personas, y deberían ser equipos que no trabajasen juntos por más de tres o cuatro años, para conseguir que no se formen roles en los que algunos miembros entren en una zona de confort que les haga ser menos productivos o estar menos motivados.

4.2.3 Informe preliminar del proyecto

Es frecuente tener una cierta ansiedad por comenzar la parte técnica, pero antes se debe hacer un estudio inicial de la propuesta, comprobar si ya hay trabajos semejantes realizados dentro de la propia empresa. Revisar el mercado y comprobar que tecnologías

se están empleando para estos trabajos. Consultar a especialistas y escuchar sus recomendaciones.

Valorar si el proyecto es necesario, si no existe ya algo que cubra esa necesidad, o si el proyecto no va a ser rentable para la empresa.

Tras esto realizar un breve documento donde se explique el alcance inicial del proyecto. Incluyendo un análisis preliminar de los riesgos, los principales stakeholders, el plan de comunicaciones, una estimación inicial de plazo y coste, etc.

El anexo 3 se corresponde con una plantilla que se podría usar a modo de checklist, para comprobar que se han tenido en cuenta los detalles del proyecto.

4.2.4 Registro de la documentación

Se crea una estructura donde se pueda almacenar la documentación referida a este proyecto.

Es importante definir un estándar de nomenclatura, para que luego resulte más sencillo ordenarla y buscarla. Un ejemplo es comenzar todos los documentos referidos al proyecto:

NombreEmpresa.Proyecto.TituloDocumento_FechaVersion

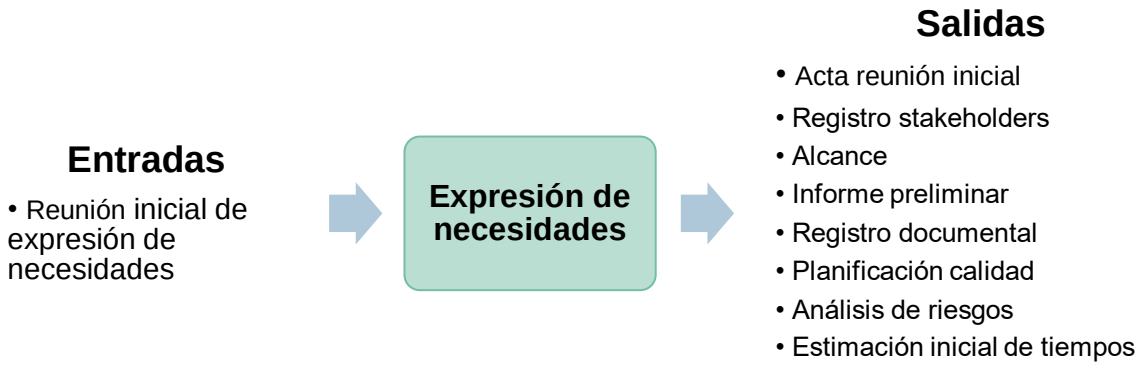
4.2.5 Planificación de la calidad

Se deberá dejar constancia de los controles de calidad, las situaciones y las pruebas que el software debe superar para ser aceptado.

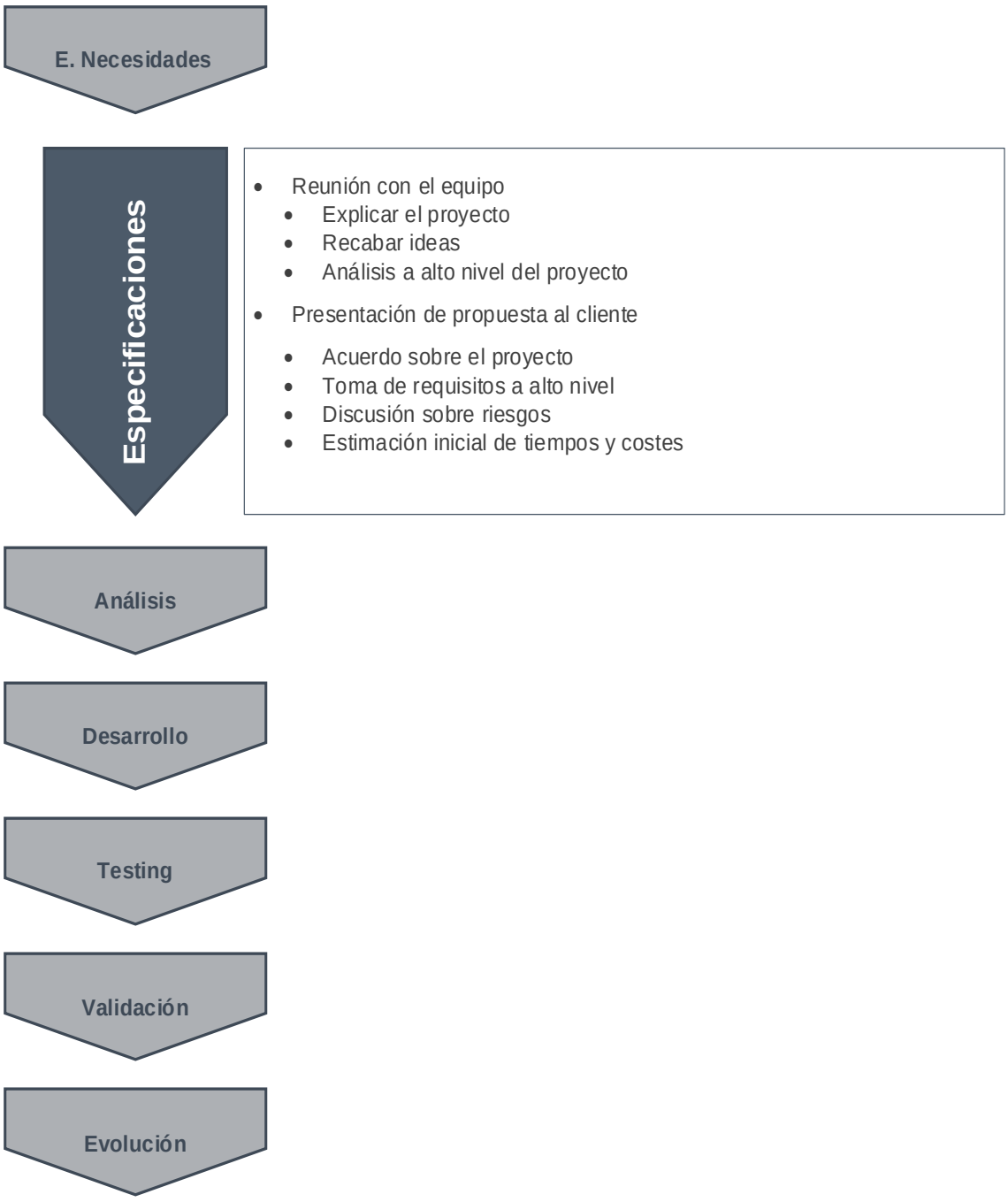
En este documento se añadirá una guía de estilos de desarrollo de código que deberán seguir los desarrolladores. Esta guía incluirá estilos de codificación, patrones de diseño si los hubiese, y procedimientos de pruebas que deberá pasar todo entregable.

Se debe dejar constancia de las herramientas de control de estilo de código usada, con las reglas y el nivel de aceptabilidad dado para el proyecto.

Por último se definirán las reglas para el control del código fuente y la estrategia a seguir para el mantenimiento de versiones y las subidas a producción.


Entradas
• Reunión inicial de expresión de necesidades
Expresión de necesidades
Salidas
• Acta reunión inicial
• Registro stakeholders
• Alcance
• Informe preliminar
• Registro documental
• Planificación calidad
• Análisis de riesgos
• Estimación inicial de tiempos

4.3 **Especificaciones**



Una vez se pone en marcha el proyecto y se cierra la parte anterior. Se deberá mantener una reunión en la que participe el responsable en el cliente y los miembros del equipo de desarrollo.

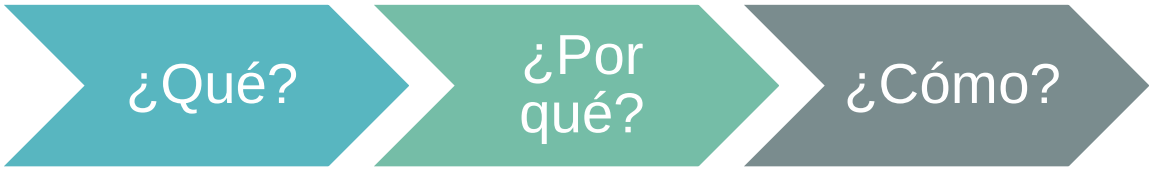
Es importante evitar "*yo te dije, tú me dijiste*", por eso es vital realizar un acta precisa que contenga toda la información que se habló para evitar estos problemas, y también para que sirva a modo de documentación del proyecto.

4.3.1 Reunión inicial con el equipo de trabajo

En una primera reunión el director del proyecto expondrá a los miembros del equipo el alcance del proyecto, y se debatirá entre todos hasta apuntalar todas las dudas al respecto, y tomar nota de las distintas alternativas.

Puede ser necesario una investigación previa sobre diferentes partes del proyecto un tanto más complejas, lo que podría dar lugar a una segunda reunión de equipo.

Esta reunión debe cubrir:



¿Qué?, es lo que necesita el cliente, indica la necesidad que nos expresó.

¿Por qué?, De esta pregunta pueden surgir alternativas, porque al mejor para cubrir la necesidad que tiene el cliente, existen otras alternativas que le pueden ser más beneficiosas, por eso es importante que los miembros del equipo entiendan el por qué el cliente tiene esa necesidad. Se puede dar el caso de que algún especialista en el equipo haya trabajo en proyectos similares donde se haya cubierto la necesidad de otra manera, o que esté al tanto de nuevas tecnologías que cubran lo mismo de forma más eficiente.

¿Cómo?, en esta parte se debatirá que tecnologías emplear, incluso en que fases se podría dividir el proyecto, y se concretará una propuesta hacia el cliente.

En algunos proyectos de innovación, se podrían utilizar técnicas de Design Thinking (Dinngo, 2018) "*Es una disciplina que usa la sensibilidad y métodos de los diseñadores para hacer coincidir las necesidades de las personas con lo que es tecnológicamente factible y con lo que una estrategia viable de negocios puede convertir en valor para el cliente, así como en una gran oportunidad para el mercado*"

Para ello se siguen unos procesos que fomentan la creatividad y fomentan el trabajo en equipo además de forma lúdica en forma de ideas innovadoras y viables.

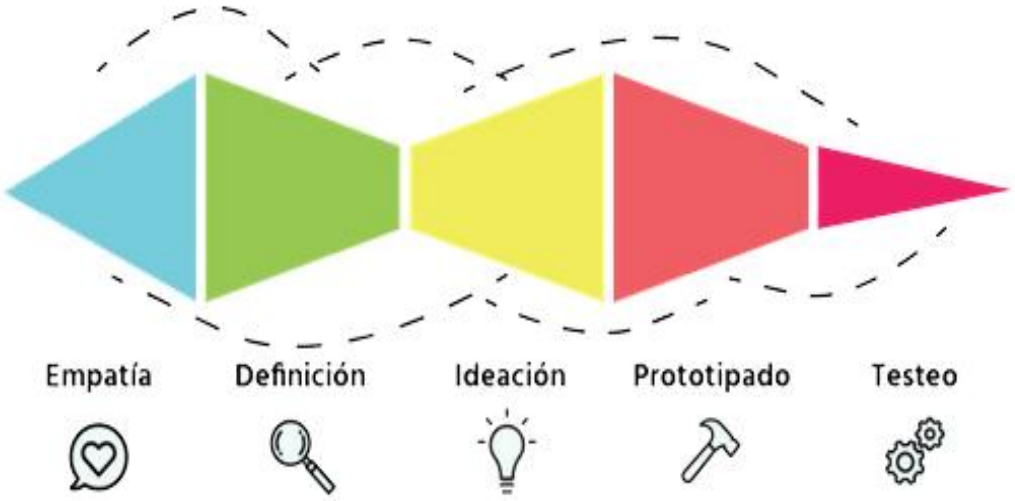


Ilustración 4.2. Fases Design Thinking (Dinngo, 2018)

Empatía. Se debe conocer profundamente la necesidad del cliente y su entorno. Se deberá ser muy conscientes de la realidad de los clientes para buscar soluciones a esas necesidades.

Definición. En esta etapa se criba la información recopilada en la fase de empatía y se queda con lo que aporta valor. Se debe identificar problemas cuyas soluciones son la clave para la obtención de un resultado innovador.

Ideación. En esta etapa el objetivo es la generación de un sinfín de opciones. Las actividades favorecen el pensamiento expansivo, se debe eliminar los juicios de valor.

Prototipado. En esta fase se ha de volver las ideas a la realidad. Construir prototipos hace las ideas palpables y ayuda a visualizar las posibles soluciones, poniendo de manifiesto elementos que debemos mejorar o refinar antes del resultado final.

Testeo. Se prueban los prototipos con los usuarios implicados, es crucial y ayuda a identificar mejoras significativas, fallos a resolver, carencias. Es una fase de evolución de la idea.

4.3.2 Reunión en cliente para presentar la propuesta de proyecto

Una vez que se tiene clara la necesidad del cliente, y que se ha aclarado las dudas, diseñado y enfocado una posible solución. Se debe hacer una reunión con el cliente, donde al igual que en la reunión inicial, acudan los involucrados. Se presenta la solución y se explican los detalles, y como se llevaría a cabo.

Es posible que en esta reunión afloren carencias o problemas que no se hayan identificado en la reunión inicial, en ese caso. Se deberá preparar una nueva reunión donde ya se aproxime a una solución satisfactoria para el cliente.

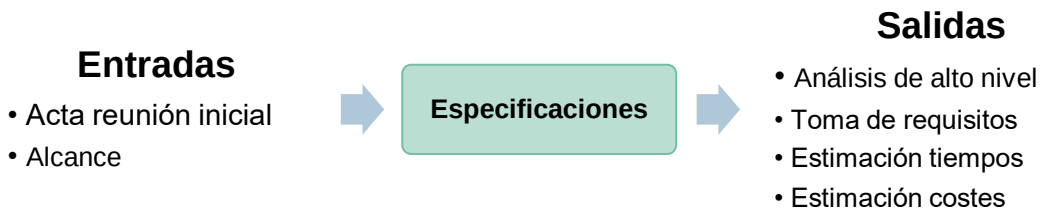
Una vez que se llega a este punto, se deben recoger todas las especificaciones y requisitos a tener en cuenta en el desarrollo a alto nivel, como por ejemplo si van a ser pantallas para Windows, si es necesario contar con aplicación móvil, que arquitectura va a tener la aplicación, etc.

En el anexo 4, se muestra un ejemplo de plantilla para esta toma de requisitos (Sevilla, s.f.).

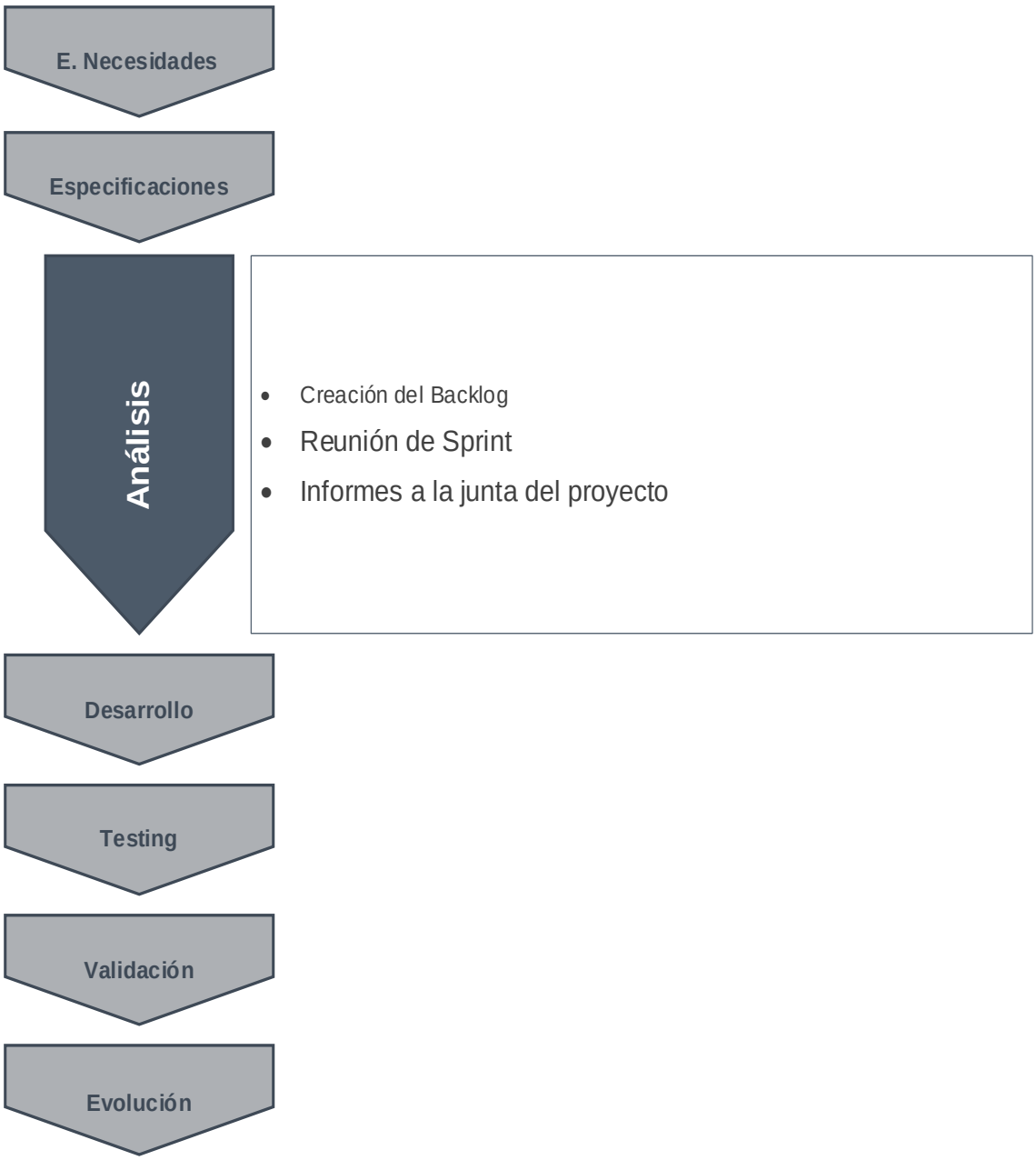
Se creará un acta de la reunión de formato similar al visto en el anexo 1, donde además se incluirá:

- Los requisitos.
- Un análisis del riesgo y qué medidas tomar para mitigarlo.
- Una estimación de tiempo y coste.

Más tarde, una vez analizado todo el proyecto en detalle, se conformarán las historias de usuario junto con los especialistas y el cliente.



4.4 **Análisis**



En el análisis ya se parte de un objetivo claro, y en proyectos grandes, la subdivisión del proyecto en fases. Por lo que se comienza por la fase acordada con el cliente.

4.4.1 Creación de las historias de usuario

Se deben reunir el cliente y el equipo de desarrollo para que el cliente explique cada historia de usuario y se vayan conformando. El equipo tomará notas, siguiendo la regla del 80/20, 80 escuchando y 20 preguntando.

El resultado final de esta reunión deberá ser una pila de backlog con historias de usuario como las de la plantilla del anexo 5.

4.4.2 Reunión de sprint

Una vez que se cuenta con el backlog inicial para la primera fase, se realiza la reunión de sprint a la que deberá asistir el representante del cliente. En caso de que el cliente no esté presente, el jefe de proyecto hará las veces de scrum máster y se pondrá en el lugar del cliente como máximo conocedor de las necesidades de este.

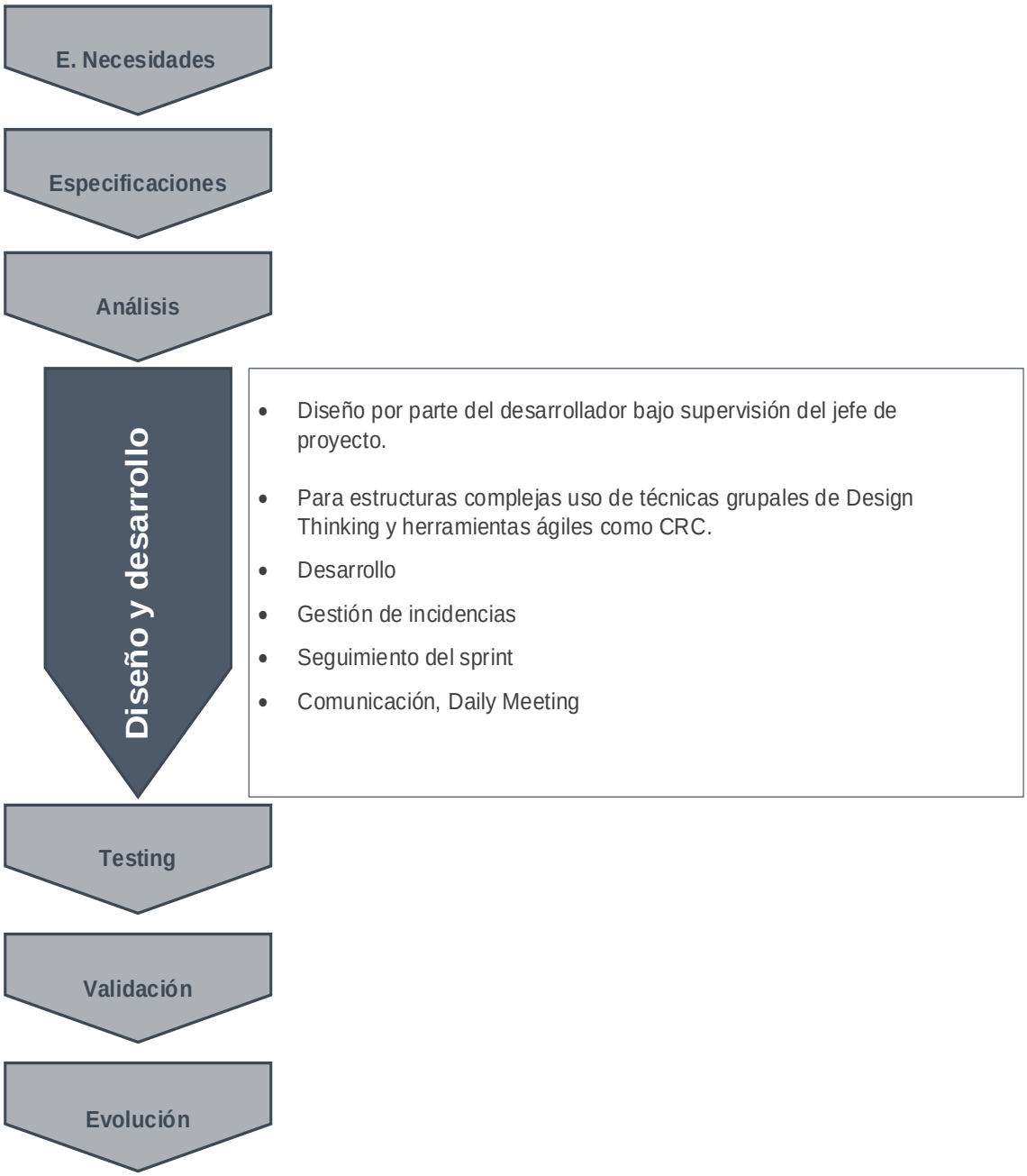
Con el backlog priorizado, se irán explicando las historias de usuario hasta que a los miembros del equipo de desarrollo les quede totalmente claro la funcionalidad a desarrollar.

En un Scrum puro, el equipo es multidisciplinar, y autogestionado. En empresas de outsourcing dedicadas al desarrollo a bajo coste, esto no suele suceder, los miembros del equipo suelen ser gente joven con poca experiencia, por lo que será el jefe de proyecto quien deberá conocer bien la tecnología y ser consciente de las posibles dificultades que pueda haber en el desarrollo, así como las peculiaridades de los miembros de su equipo. Por esto será éste quien reparta las tareas y estime los plazos. La intención del jefe de proyecto en todo caso siempre ha de ser la de formar un equipo que con el tiempo pueda llegar a ser autogestionado.

4.4.3 Informes a la junta del proyecto

En proyectos pequeños, para mantener la agilidad documental y del proceso, los informes correspondientes a las etapas intermedias del proyecto se pueden dar de forma verbal o por un simple email. En las etapas de inicio y fin, deberá ser un informe formal que se almacenará junto al resto de documentación del proyecto.

4.5 **Diseño y desarrollo**



4.5.1 Diseño

Esta parte corre a cargo del desarrollador responsable de la tarea, aunque siempre supervisada por el jefe de proyecto, para que él, que tiene una visión global del proyecto, decida los temas relacionados con la arquitectura del mismo.

En esta etapa se pueden generar documentos de tipo diagrama de clases, mapeo de comunicaciones, etc. Estos documentos tienen que ser correctamente registrados y almacenados en la estructura de la documentación.

La documentación del código quedará registrada en el control de versiones.

En diseños complejos se pueden usar técnicas como las tarjetas CRC (Clase, Responsabilidades y Colaboradores) (agilesurfers.es, 2018). Esta actividad es una especie de juego de rol que se utiliza para esbozar rápidamente las ideas para el diseño de un sistema orientado a objetos.

Dos o más miembros del equipo, escriben en fichas los nombres de las principales clases involucradas con este formato:

Nombre de la clase	
Responsabilidades	Colaboradores
-	-
-	-
-	-

Tabla 4.1. Formato tarjeta CRC (autor)

Las responsabilidades son las características de la clase y sus colaboradores las clases que dependen de ellas para llevar a cabo sus propias responsabilidades.

Ahora se valida cada idea de diseño, para ello cada desarrollador asume el rol de una o más clases y se da vida al escenario.

Por ejemplo:

- *"Hola, controlador de autenticación, soy una solicitud web y me gustaría conocer el contenido de este recurso.*
- *"Muy bien dame tus credenciales, junto con el nombre de la operación a realizar"*

Esta herramienta fue creada por Kent Beck y Ward Cunningham, hay que recordar que Kent Beck es uno de los firmantes del manifiesto ágil.

Es una herramienta que fomenta la comunicación y el aprendizaje en conjunto, ayuda a crear equipos multidisciplinares en que cada miembro tenga una forma de ver la estructura global de forma similar, pero puede dar lugar a roces y ser un poco lenta, por lo que es aconsejable usarla sólo para estructuras complejas o innovadoras que merezca la pena usar un tiempo en su diseño.

Para rellenar las tarjetas se puede usar la lluvia de ideas de manera que:

- En un principio todas las ideas son válidas
- Pensar rápido, se razona después
- Cada miembro tendrá un turno sin presiones

Es de vital importancia hacer diseños usando patrones de diseño que están ya contrastados y evitan problemas posteriores.

Tendrá que ser muy modular, de forma que se puedan reaprovechar los diferentes módulos en otros diseños, esto hará la aplicación mucho más mantenible y robusta, permitiendo cambios en los distintos módulos sin que estos afecten al resto. Para ello se debe utilizar interfaces que abstraigan el código.

Los diseños pueden ser ágiles, no es necesario definir el diseño completo al inicio. Se puede partir de un diseño general e ir construyéndolo a medida que avanza el desarrollo, de esta manera se adaptará a cambios y evoluciones.

Separar completamente el diseño de la codificación tiene sus riesgos y siempre es mejor contar con profesionales que puedan diseñar y codificar.

El diseño estético de las interfaces es algo que no se debe descuidar, por un lado es la imagen de presentación de la empresa en el cliente, y por otro lado una interface accesible y con los elementos bien distribuidos, contribuye a un mejor aprovechamiento del producto por parte del cliente y favorece el éxito del proyecto.

Se debe realizar un boceto inicial que sirva para detectar posibles errores en las especificaciones y evite problemas en el desarrollo posterior de la arquitectura.

Se debe supervisar durante el desarrollo para asegurarse que cumple con los cambios que se pudiesen haber introducido.

Y al final, a la hora de recoger el feedback del cliente en cuanto a usabilidad del producto.

4.5.2 Desarrollo

Durante la fase de desarrollo si surge alguna incidencia, como por ejemplo que alguna parte del análisis se haya considerado erróneamente y sea necesario replantearla, esto se comunicará al representante en cliente, y una vez replanteada, se actualizará la historia de usuario y se seguirá el ciclo normal.

De ser una incidencia compleja, se deberá clasificar en función de la situación:

- Incidencia que afecta que genera dependencia al resto de tareas del sprint: Se volverá a convocar una reunión de Sprint, donde se volverá a analizar esa parte, y se reestructurará la lista de sprint con las nuevas tareas.
- Incidencia que no genera dependencia, se puede sustituir por otra historia de usuario y dejar el análisis de esta para la reunión de sprint siguiente.

Es necesario registrar la comunicación con el cliente donde se acuerdan los cambios en el planteamiento. Bien vía email, registrando dichos emails, y las versiones de la historia de usuario, o de existir una reunión, un acta con dicha reunión que se enviará al cliente para que quede constancia.

4.5.3 Seguimiento del sprint

Diariamente los desarrolladores deberán rellenar una hoja para ver el gráfico de burn-down.

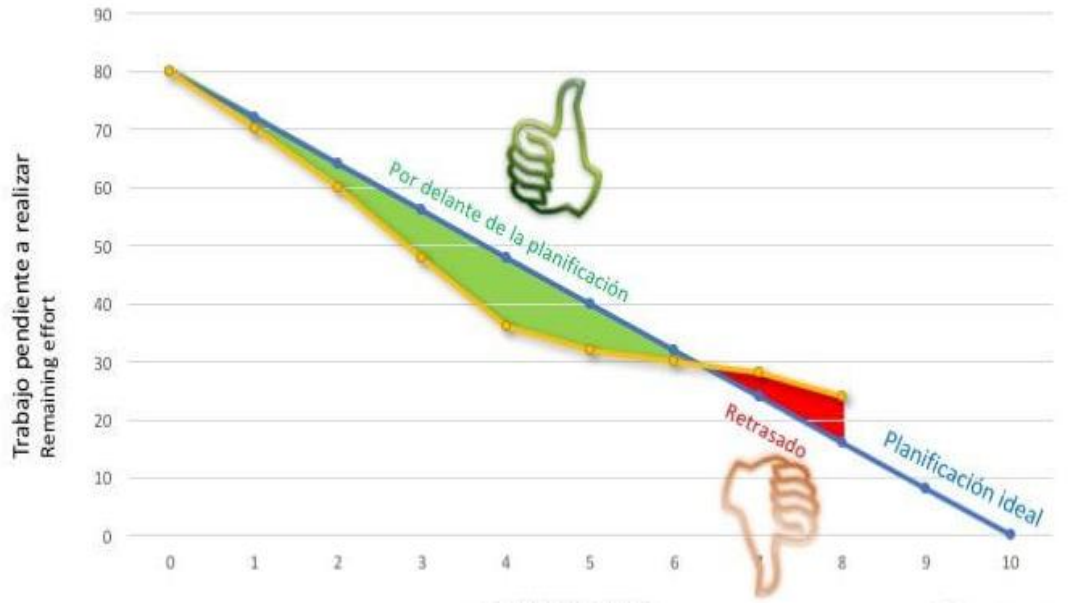


Ilustración 4.3. Gráfico de Burn Down (Alexander Menzinsky, 2016)

En el anexo 6 se puede ver una plantilla tipo, para Sprints de 2 semanas de duración.

Se pueden dar situaciones anómalas durante el Sprint, por ejemplo:

→ El cliente (Product Owner) pide incorporar trabajo extra.

Si puede esperar, se deja para el siguiente Sprint, es lo ideal, pero si no pudiese esperar, se debe interrumpir el sprint y reiniciar uno nuevo (Pérez Á. N., s.f.).

Si es habitual que el cliente, por necesidades de su negocio, adelante tareas, o modifique el Sprint, se puede considerar esto en el propio Sprint, reservando un tiempo para dichas tareas, en previsión de estas fallas, de forma de que si esto ocurre se consumirá dicho tiempo, y si no llega a darse, al ir terminando el sprint, se adelanta una nueva tarea de la pila que no estaba introducida inicialmente.

→ El equipo identifica nuevas actividades, lo que implica más esfuerzo.

En este caso no se están añadiendo funcionalidades extra, en este caso se modifica el trabajo total del sprint incorporando el esfuerzo adicional, o bien se quitan tareas para cuadrar el esfuerzo a la duración del sprint.

→ El equipo va más rápido de lo planificado.

En este caso, se añadirán las siguientes tareas planificadas en la pila de backlog que no hayan entrado en este sprint y será un síntoma de una mala planificación en el sprint actual porque lo ideal es terminar las tareas a la vez que el sprint y lo normal es que no se puedan concluir todas y quede al menos un poco de esfuerzo para concluir alguna de las historias de usuario.

En la reunión post-sprint, se deberá analizar estas situaciones y entre todos idear soluciones para intentar que no se repitan estas situaciones anómalas o que si se repiten estén contempladas acciones para mitigarlas.

Además, en estas reuniones post-sprint se puede valorar la velocidad del equipo y discutir sobre cómo mejorarla.

Los desarrolladores rellenaran el kanban, lo ideal es que sea una pizarra física que esté visible por todo el equipo y de esa manera todos sean conscientes en todo momento de cómo está resultando el sprint. De esta forma el gestor del proyecto de una forma rápida puede visualizar el estado del proyecto e identificar posibles cuellos de botella o retrasos.

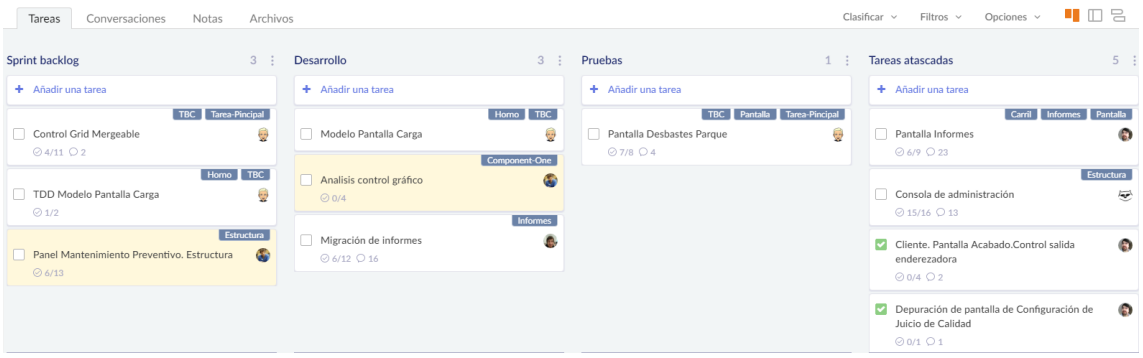


Ilustración 4.4. Ejemplo panel Redbooth con 5 pilas de tareas (autor)

4.5.4 Comunicación

Durante el desarrollo es muy importante la comunicación entre los miembros del equipo. *"La gestión es un 5% instrucción y un 95% comunicación"*. (Appelo, 2011).

Que los miembros del equipo estén al tanto de lo que hacen sus compañeros propicia (Appelo, 2011):

- Colaboraciones en forma de ayuda.
- Compartir conocimiento, lo que ayuda a la formación de equipos multidisciplinares.
- Empatías que consiguen un mayor vínculo entre los miembros del equipo.
- Equipos robustos que consiguen tirar unos por otros para llegar a un buen resultado

Para ello es importante contar con herramientas tipo chat, para que, aunque trabajen juntos, no se interrumpan, si es tipo chat, cada desarrollador podrá decidir en qué momento atiende el mensaje, de esa forma se propicia que trabajen en la tarea sin interrupciones. En las interrupciones es donde se acumulan la mayor parte de los errores de software y debe propiciarse un lugar de trabajo que las evite. Para ello se pueden adoptar ciertos procedimientos:

- Usar las primeras horas del día en que el equipo está fresco, para trabajar sin distracciones, esto quiere decir que se deben desconectar el correo y cualquier tipo de notificación. Dejando sólo el teléfono, para cosas que de verdad sean urgentes. De esta manera se logra que esas horas sean de máxima concentración y de máxima producción, realizando en ellas las tareas más complejas y relevantes en el sprint.
- No poner reuniones en las primeras horas del día, pero tampoco a última hora, a primera hora por lo indicado en el punto anterior, y a última porque el equipo ya está cansado y tiende a relajarse y no abordar la reunión con una concentración suficiente.
- Llegar a acuerdos entre los miembros del equipo de no interrupción, estableciendo protocolos para ello, de manera que se usen la herramientas digitales para comunicarse en una primera instancia, y luego si es necesario, ya juntarse para hablarlo.

Además, se hacen reuniones diarias, a media mañana, para no interrumpir esas primeras horas de máxima productividad, de unos quince minutos aproximadamente, donde de pie y situados en círculo, los desarrolladores deberán responder a:

- ¿Qué hice ayer para ayudar al equipo a alcanzar el objetivo del sprint?
- ¿Qué voy a hacer hoy?
- ¿Estoy teniendo algún problema?

Si existe un Kanban físico, lo ideal es hacerlo delante del Kanban y aprovechar a actualizarlo.

Ilustración 4.5. Daily Meeting (Oufaska, cafe-agil.com, 2018)

Los beneficios del Daily son: (Oufaska, cafe-agil.com, 2018)

- Se elimina la necesidad de otras reuniones innecesarias.
- Se identifican problemas en el sprint, para atajarlos lo antes posible.
- Promueve la toma rápida de decisiones
- Mejora el nivel de conocimiento del equipo
- Mejora la colaboración

Se debe realizar en un tiempo máximo de quince minutos, porque esto anima a los equipos a empezar a trabajar rápidamente en los objetivos, la teoría de la motivación temporal

(Steel, 2010), demuestra que las limitaciones de tiempo son un componente fundamental en la búsqueda de la eficiencia.

Según esta teoría, la formula U=EV/ID explica la procrastinación, donde:

Motivación = EV/IR

E: Expectativa de conseguir el objetivo
V: Valor atribuido a ese objetivo
I: Impulsividad

Cuanto mayor es la expectativa o creencia de conseguir algo, y más valor le atribuimos a ese algo, mayor es la motivación.

Cuanto más impulsivos somos, y menos disciplinados, junto con lo que falta para llegar al objetivo (el retraso), reduce la motivación.

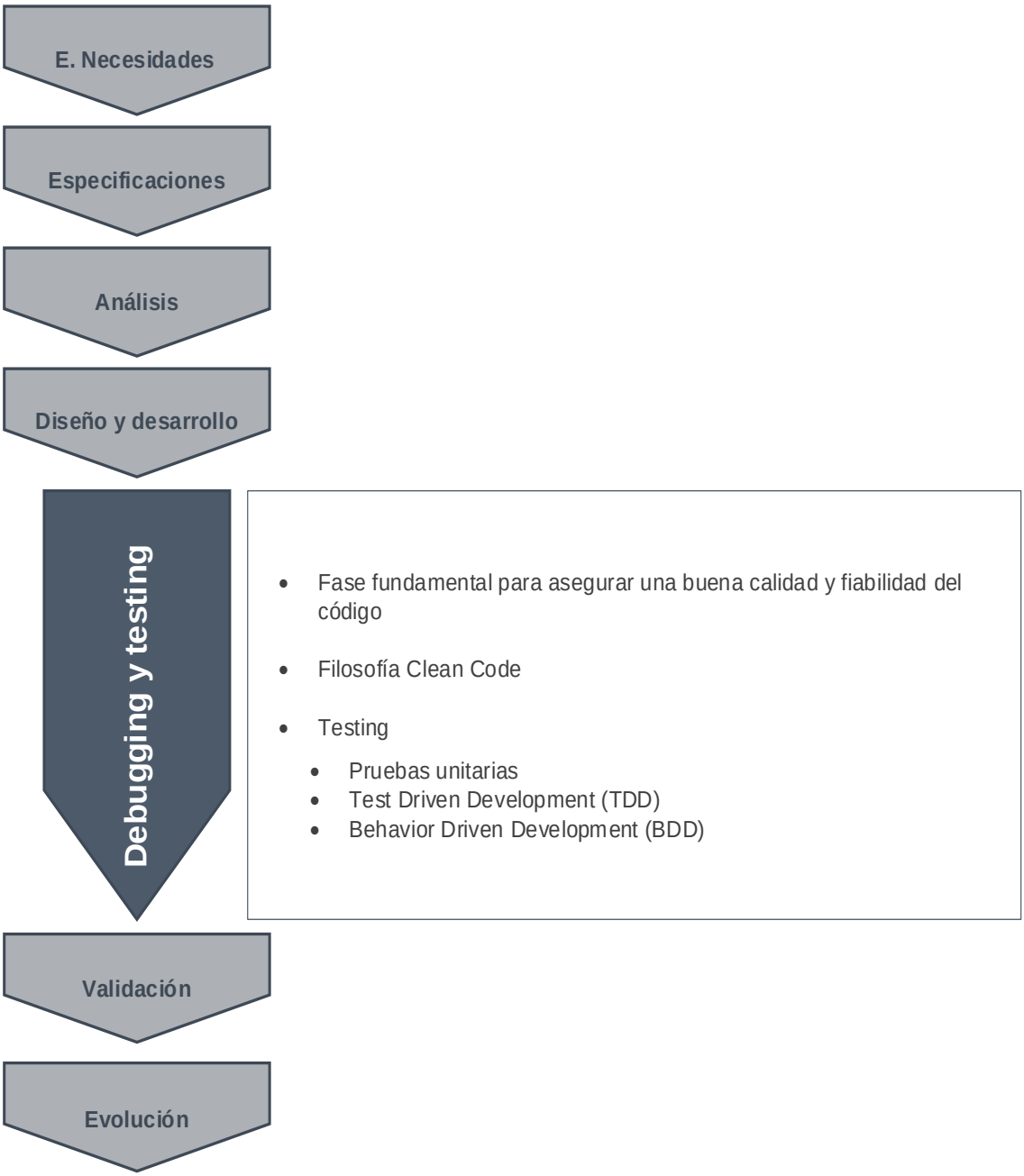
Si el Daily dura más de quince minutos hay que replantearse que algo no se está haciendo bien, seguramente se traten cosas que no tengan que ver con el objetivo del sprint.

Es importante dividir mucho las tareas del sprint de forma, que nos ayude a ir cerrando etapas (motivación), y que no se vuelva muy repetitiva la conversación del Daily.

Si surgió un tema que va a exceder el tiempo del Daily, se debe posponer y tratar ese tema en una reunión posterior con las personas implicadas.

Hay que tener en cuenta que no es una reunión de reporte, en el que cada uno dice lo que hizo y lo que hará, sino una reunión de equipo en que se habla sobre el objetivo del sprint y como abordarlo, en rugby (de donde viene el nombre de Scrum), sería como cuando en círculo se abrazan los jugadores a hablar cosas como: *"A mí me está dando problemas ese defensa", "Yo me encargo de eso", "Nuestro juego no está funcionando, sorprendámoslos haciendo un pase…"*

4.6 **Debugging y Testing**



Como se demuestra a lo largo de este documento, la parte diferencial de desarrollar software industrial es la alta fiabilidad y calidad que debe tener su código. Es por esto que esta fase es fundamental y hay que tomarla muy en serio.

Una vez que el desarrollador termina de codificar, deberá gastar un tiempo en refactorizar bien su código, asegurándose que su desarrollo cumple las premisas del código limpio (Martin, 2012), este autor, Robert C. Martín (conocido como el Tío Bob) es también uno de los firmantes del manifiesto ágil.

"Cualquiera que sea capaz de escribir programas en un lenguaje de programación es un "programador"; la diferencia entre un "programador" y un "ingeniero de software" radica en las practicas, la estandarización y el cuidado que damos a nuestro trabajo, no solo para nuestro consumo sino el de los demás. Es decir, la principal diferencia radica en el profesionalidad" (Martin, 2012).

4.6.1 Código limpio

Esta filosofía pretende que el código sea más fácil de leer, mantener, extender y menos propenso a errores.

Según esto hay doce premisas que se deben de cumplir a la hora del desarrollo (Martin, 2012):

4.6.1.1 Utilizar nombres con significado

El código debe poder leerse como si de un libro se tratase, para ello se debe buscar nombres para las clases, métodos, propiedades, que den sentido a lo que representan aunque los nombres sean largos, con los editores de código actual, el que sean largos no es un problema. Lo importante es explicar claramente su función

4.6.1.2 Unidades de código pequeñas

Las clases y métodos cortos simplifican la compresión del código y lo hacen más mantenible. Si hacen objetos demasiado grandes, se tardará muchísimo en entender su funcionalidad por lo que sus mantenimientos se harán muy pesados.

Es habitual limitar el número de líneas de clases y funciones con el fin de obligarnos a cumplir ese punto, como norma general 10-20 líneas por método y 500 líneas por clase.

No superar más de un nivel de anidamiento en los if/else.

4.6.1.3 Una única responsabilidad por unidad de código

Principio de responsabilidad única (uno de los cinco principios **SOLID**). Esto es interesante porque si no lo hacemos nos podemos encontrar con los siguientes problemas:

- Clase propensa a cambios, al tener más de una responsabilidad, aumentan exponencialmente el número de razones por las que se puede dar la necesidad de modificar la clase.
- Nos cuesta más testearla, los tests serán muy largos y poco eficaces.
- El código crecerá de forma desordenada, y será más complicado añadir nueva funcionalidad.

4.6.1.4 Funciones con un máximo de tres parámetros

Los parámetros añaden complejidad y carga conceptual que dificultan la lectura, si fuese necesario pasar varios argumentos, es recomendable crear estructuras. Además dificultan mucho el testing, porque cuantos más parámetros más combinaciones.

4.6.1.5 Principio DRY (Don't Repeat Yourself)

Procurar no repetir código, las razones fundamentales para esto:

- Nos obliga a testear lo mismo varias veces
- Si se detecta un bug en un bloque, tendremos que acordarnos de cambiarlo en el resto
- Amplían la cantidad de código a mantener

4.6.1.6 Evitar utilizar comentarios siempre que sea posible

Si se tiene que usar un comentario, es porque no se ha conseguido hacer un código lo suficientemente claro. El código debe ser auto explicativo.

Existe el inconveniente, de que luego se modifica ese código y muchas veces no el comentario. Además, el cerebro del desarrollador suele ignorar los comentarios. En cualquier caso, en situaciones en las que no se sea capaz a simplificar el código, si se debe añadir un comentario.

4.6.1.7 Formato único en el código

Usar un estilo de código aunque no sea bueno, es mejor que no usar nada, porque este estilo dará unas pautas a seguir por todos los desarrolladores. Se pueden acordar cosas como:

- Densidad vertical: Los conceptos relacionados deberían aparecer juntos verticalmente.
- Ubicación de los componentes: por ejemplo dividir las clases en regiones, con esta secuencia: atributos -> propiedades -> constructores -> delegados -> métodos públicos -> privados.
- Número de líneas máximo por método
- Número de líneas máximo por clase
- Número de caracteres máximo por línea horizontal, normalmente 120 caracteres

4.6.1.8 Abstracción de los datos

Uso de interfaces, para lograr exponer sólo lo necesario de la clase, e independizarla del resto de la estructura, lo que facilitará mucho luego su mantenimiento.

4.6.1.9 Ley de Demeter

El objeto no debería conocer las entrañas de otros objetos con los que interactúa. Es un mecanismo de detección de acoplamiento. Para solicitar algo a otro objeto, se debería tener la capacidad de pedírselo directamente, sin tener que navegar por su estructura.

Por ejemplo:

```
getX().getY().getZ().hazEsto();
```

El problema es que:

- Necesitamos conocer la estructura de las clases para poder hacer esto.
- El código será propenso a modificaciones, porque se depende de varias clases propensas a modificaciones.

La solución no es ocultar el problema creando métodos que oculten esto, sino restructurar el código delegando en el responsable la ejecución de esta tarea.

4.6.1.10 Lanzar excepciones en lugar de códigos de retorno

Se consigue:

- No es necesario recordar operar con ese código de error, la excepción lo hace por ti.
- Se separa la lógica del "camino feliz" de la de errores.
- Lo idóneo son excepciones propias que den un significado más semántico al error de un vistazo, y proveerlas de mensajes de error claros y descriptivos.

4.6.1.11 Establece fronteras

Es útil para abstraernos del código que no controlamos, de tal forma que podamos acotar la interacción con él y que permita sustituirlo sin problemas en caso de necesidad.

4.6.1.12 Test unitarios

Los tests nos ayudan a definir y validar la funcionalidad del software que se está implementando.

Para cumplir muchas de estos patrones se puede configurar los editores como Visual Studio o usar herramientas como Sonar Lint, donde se configuran algunas de estas reglas y después estas herramientas analizan el código y descubren los lugares donde no se están cumpliendo, facilitando la refactorización de este código.

4.6.2 Testing

Para hacer el testeo del código y comprobar que se cumple con las especificaciones propuestas, con cualquier combinación de uso y se controla vía código todas las situaciones, podemos distinguir entre:

4.6.2.1 Pruebas unitarias

Se centran en una única unidad de código. Es una prueba sencilla, rápida de escribir y ejecutar. Son útiles sobre todo al tener que cambiar el código, porque al ejecutarlas comprobamos que se sigue comportando correctamente.

No debe tener dependencias, no debe ejecutar framework de aplicación ni código externo. Se tiene que probar el componente aislando de los demás. Los tipos más habituales de pruebas son:

→ **Pruebas unitarias**: Se prueba una única pieza de código.

→ **Pruebas de integración**: Son pruebas que se ejecutan después de las unitarias, para probar conjuntos de software más amplios.

→ **Pruebas funcionales o de aceptación**: Prueban el conjunto de la aplicación.

4.6.2.2 Test Driven Development (TDD)

Es una técnica de diseño e implementación de software incluida dentro de la metodología XP. Sigue estos pasos:

1. Se escribe la prueba que recoge nuestros requisitos
2. Ejecutamos la prueba. Tendrá que fallar ya que todavía no desarrollamos el código.
3. Se escribe el código necesario para que la prueba pase.
4. Se vuelve a ejecutar y esta vez corre con éxito.

De esta forma nos estamos asegurando que se cumple con los requisitos, porque es lo primero que validamos. Además, aunque se tarda más en el desarrollo, es un tiempo que se recupera porque además de ayudarnos a validar el código, este patrón sirve para:

- Tener una documentación del código, ya que si se usa una nomenclatura adecuada, al leer las pruebas, cualquier programador es consciente de la funcionalidad de ese código.

  Una nomenclatura muy usada es la *WhenItShould*.

  Given/When: condiciones, configuración o input inicial

  It: unidad o sujeto testeado

- Facilitan el mantenimiento de código porque protegen de cambios erróneos.

- Evitan rollbacks en subidas de código que provoquen errores
- Ayudan a encontrar inconsistencias en los requisitos
- Ayudan a especificar comportamientos
- Ayudan a refactorizar para mejorar la calidad de nuestro código
- A medio/largo plazo aumentan la productividad.
- Simplifican la integración de sistemas pues permiten llegar a la integración con la seguridad que los módulos independientes funcionan correctamente.

Las TDD tienen una cobertura del 90-100% lo que implica que añadir nuevas funcionalidades y mantenerlas es sencillo. Además es recomendable usarlas junto a las pruebas unitarias. Tienen una curva de aprendizaje lenta, pero una vez arrancado es un proceso ágil y rápido. Es un inversión a largo plazo, porque donde de verdad aporta valor es en el mantenimiento del código.

4.6.2.3 Behavior Driven Development (BDD)

Desarrollo guiado por el comportamiento. Es una evolución del TDD, son pruebas que verifican que el comportamiento es correcto desde el punto de vista del negocio (Delgado, 2018). Se parte de historias de usuario, y se usa un lenguaje especifico de Gherkin donde se puede programar directamente los criterios de aceptación de dicha prueba.

Gherkin es un lenguaje común, que no requiere conocimientos de programación, pero que lo puede comprender también un programa. Para ello se crea un archivo .feature donde se especifica:

Feature: nombre de la funcionalidad que vamos a probar.

Scenario: Uno por cada prueba.

Given: Precondiciones de dicho escenario

When: Se especifican las acciones que se van a ejecutar

Then: El resultado esperado

Puede haber por cada archivo un *feature* con varios escenarios para la prueba.

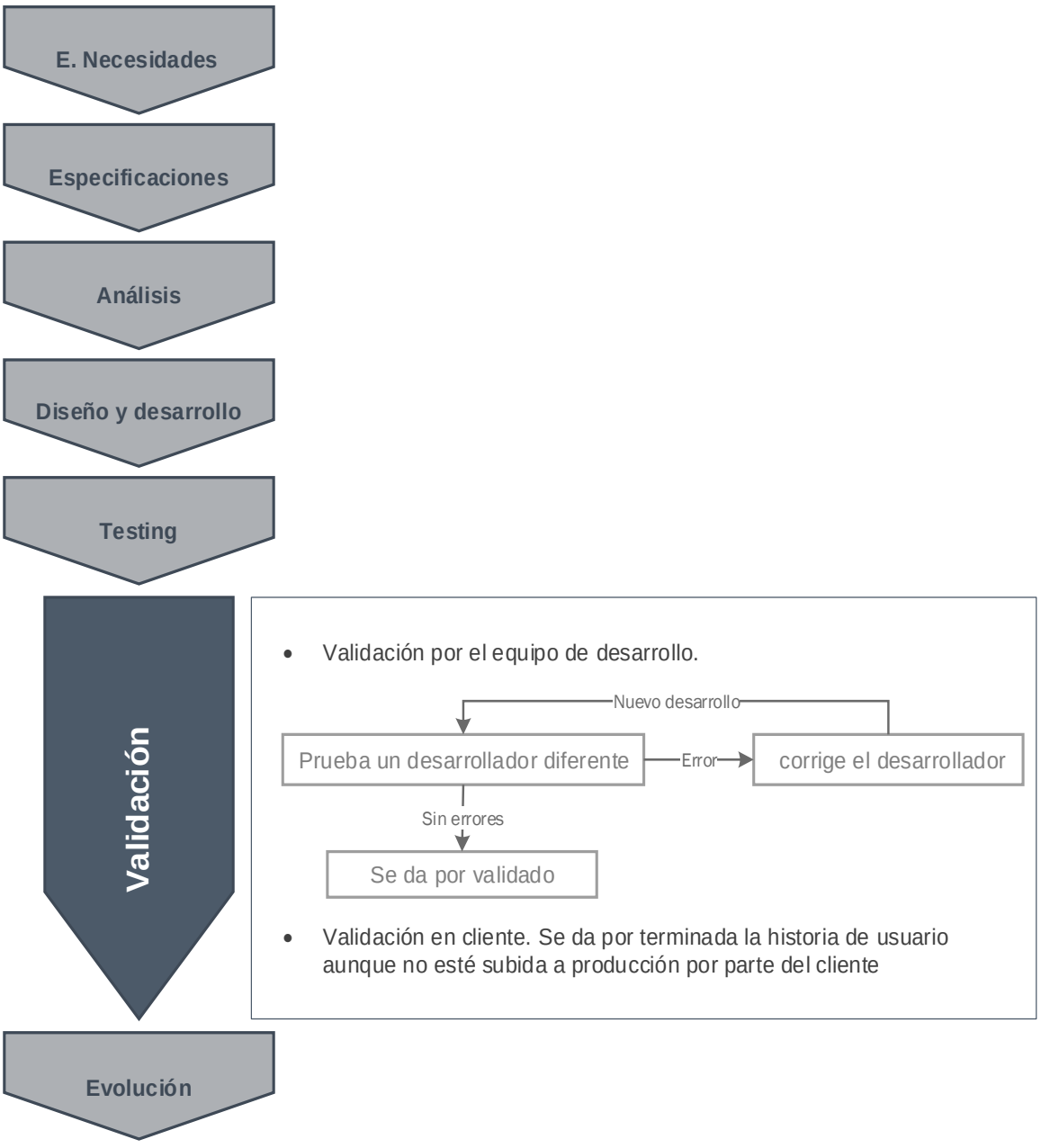
Ejemplo:

Feature: Serve coffee
Coffee should not be served until paid for
Coffee should not be served until the button has been pressed
If there is no coffee left then money should be refunded

Scenario: Buy last coffee
Given there are 1 coffees left in the machine
And I have deposited 1€

En conclusión las pruebas unitarias te dicen que testear, la TDD cuándo y las BDD cómo.

4.7 **Validación**



4.7.1 Validación por el equipo de desarrollo

Una vez que el desarrollador dar por finalizada la historia de usuario, si se trata de una interfaz de usuario, otro miembro del equipo o incluso de otro equipo, deberá probar que cumple con todos los requisitos, además de las reglas de estilo que tenga la aplicación y las normas de usabilidad.

Para ello deberá dejar un registro, bien en alguna herramienta que tenga la empresa para esto, o a través de una plantilla como por ejemplo la del anexo 7.

En esta plantilla se comprueba si la funcionalidad indicada, más todas las normas de la empresa o las fijadas por el cliente que sean comunes a todo el proyecto cumplen o no.

Tras esto, si se cumplen todas las especificaciones, ya estará listo para entregar al cliente, en caso contrario se devolvería al desarrollador que:

1. Reparará las funcionalidades erróneas.
2. Volverá a correr todos los test unitarios
3. Lo entregaría de nuevo al probador.

El probador deberá volver a cubrir la plantilla entera probando de nuevo todo, incluido las funcionalidades que anteriormente habían pasado las pruebas, porque se puede dar el caso que al reparar una funcionalidad errónea se modificase una funcionalidad que si estuviese correcta.

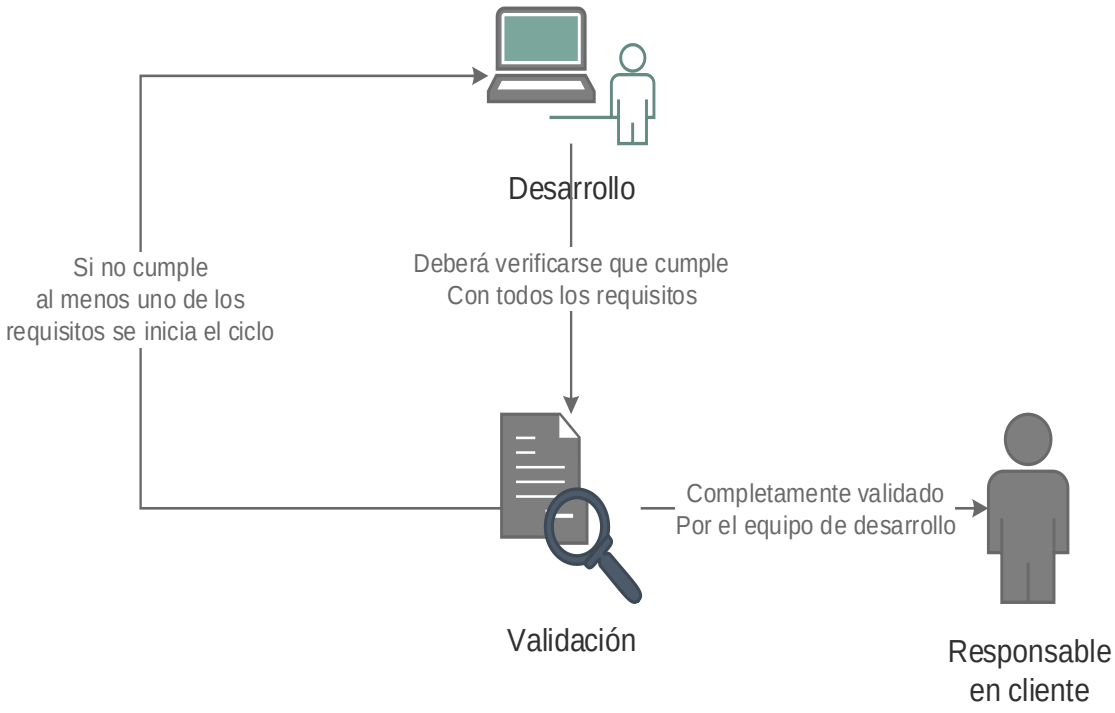


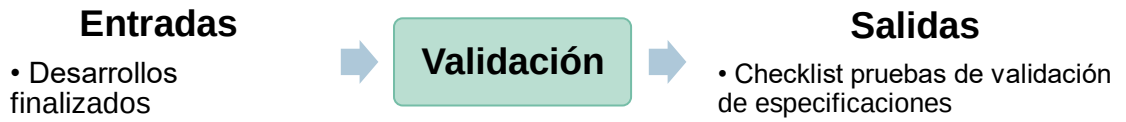
Ilustración 4.6. Validación por parte del equipo de desarrollo (autor)

4.7.2 Validación en cliente

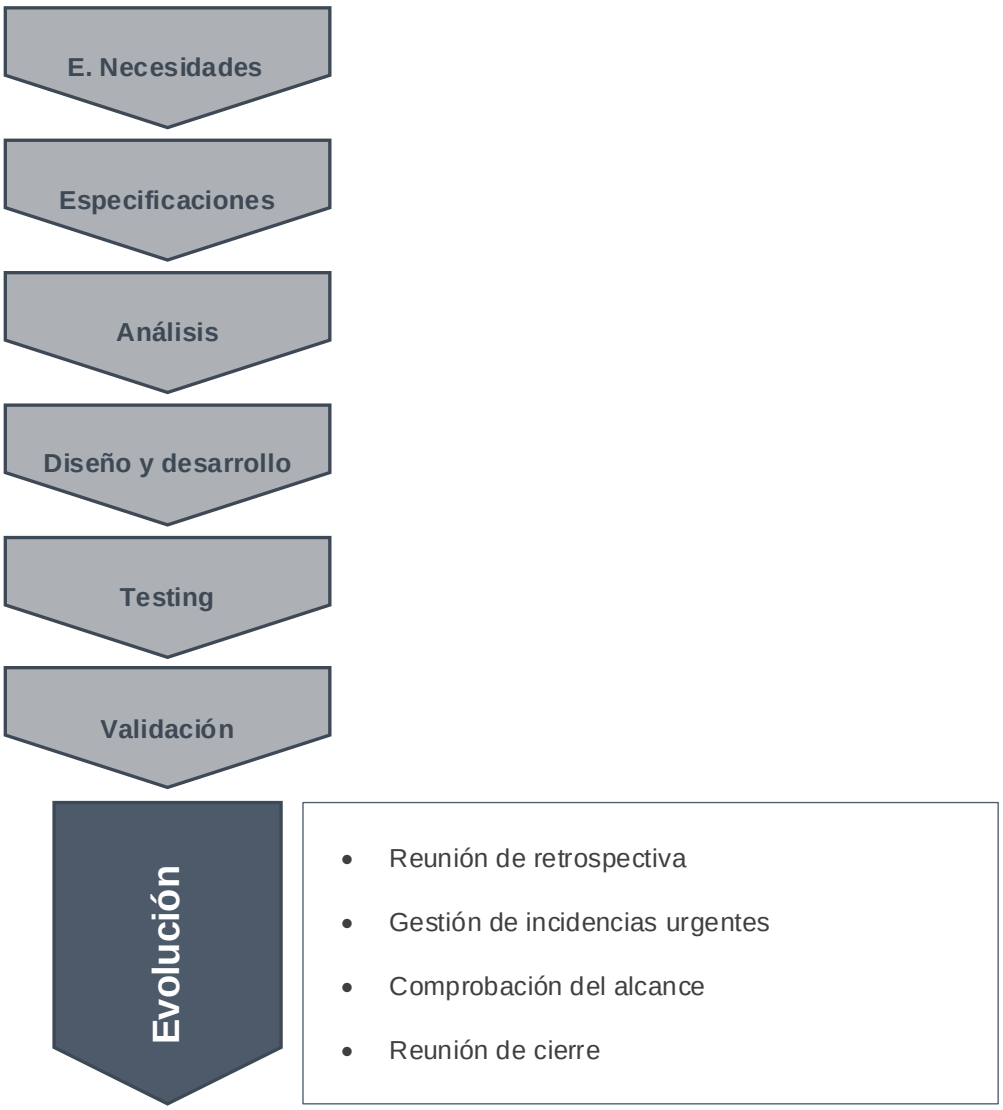
El jefe de proyecto informará al responsable en cliente, de que el producto está ya en integración listo para subir a producción.

El cliente deberá verificar que el producto cumple con los requisitos solicitados, y podrá mostrarlo al resto de implicados en cliente. Por bien que se hayan desarrollado las etapas anteriores, y por mucho que se hayan cuidado los detalles en la toma de requisitos y análisis, en esta etapa lo normal es que el cliente se dé cuenta de que faltan cosas, que no se contemplaron ciertos escenarios, o se le ocurran mejoras. Estos cambios, se tratan como nuevas historias de usuario que pasarán a la pila de backlog priorizadas según el criterio del cliente y que comenzarán el ciclo de desarrollo como si de una historia nueva se tratase.

Es por esto que las tareas de sprint terminan una vez que se ponen las historias de usuario en integración y no en producción como marca la teoría, esto se debe a que esta puesta en producción no depende del equipo de desarrollo sino del cliente.



4.8 **Evolución**



Una vez que se termina la fase anterior, se volvería a realizar una nueva interacción del proyecto, con una nueva reunión de Sprint, donde se volverían a decidir qué historias de usuario abordar para seguir avanzando en el proyecto. Se puede dividir esta reunión en dos fases y en la primera fase dedicarla a hacer retrospectiva, de forma que se repasen todas las historias de usuario realizadas en el sprint anterior contemplando:

- Si se terminó según lo esperado y sin problemas.
- Si no se terminó, porque fue, ¿mala planificación de tiempo?, ¿problemas inesperados?, ¿problemas del miembro del equipo asignado?
- ¿Qué problemas surgieron que no estaban planificados?

- ¿Cómo abordar esos problemas en el futuro?
- ¿Cómo mejorar el rendimiento del grupo?
- ¿Qué riesgos empiezan a aparecer? ¿Cómo afrontarlos?
- Perspectiva general del proyecto

Es importante recoger la opinión de todos los miembros del equipo para que se mantengan involucrados en el proyecto y se sientan participes de las decisiones que adopta el grupo.

Intentar recoger el feedback del usuario final como parte de la mejora de calidad, con encuestas adaptadas al tipo de proyecto que se entregó para lograr ver donde se puede mejorar en los siguientes desarrollos o atajar posibles fallos de diseño y funcionalidad.

A la par que se siguen desarrollando partes del producto, pueden surgir problemas en las partes que están en producción, parte de estos problemas se pueden gestionar como si se tratase de un cambio y procurar introducirlo en un nuevo sprint, pero habrá otras incidencias que necesitarán de una atención inmediata al estar generando malos resultados con consecuencias para el cliente. Se debe prever estas situaciones, y a la hora de planificar el sprint, dejar un buffer de tiempo para el desarrollador más susceptible de que tenga que solucionar de urgencia un caso como este. Por ejemplo, si un desarrollador es el responsable de una historia de usuario relativa a un proceso delicado, el cual se estima, y se ha considerado así durante la gestión del riesgo, que puede dar lugar a algún tipo de problema, ese desarrollador en el siguiente Sprint, se le guardará un margen de tiempo por si se diese ese caso poder mitigar el riesgo rápida y eficientemente sin que afecte en demasía al curso normal del sprint y sobre todo, intentando que afecte lo menos posible al cliente.

El director del proyecto debe estar muy pendiente en esta fase, de comprobar el alcance del proyecto y en caso de que sea necesario proponer el cierre del proyecto.

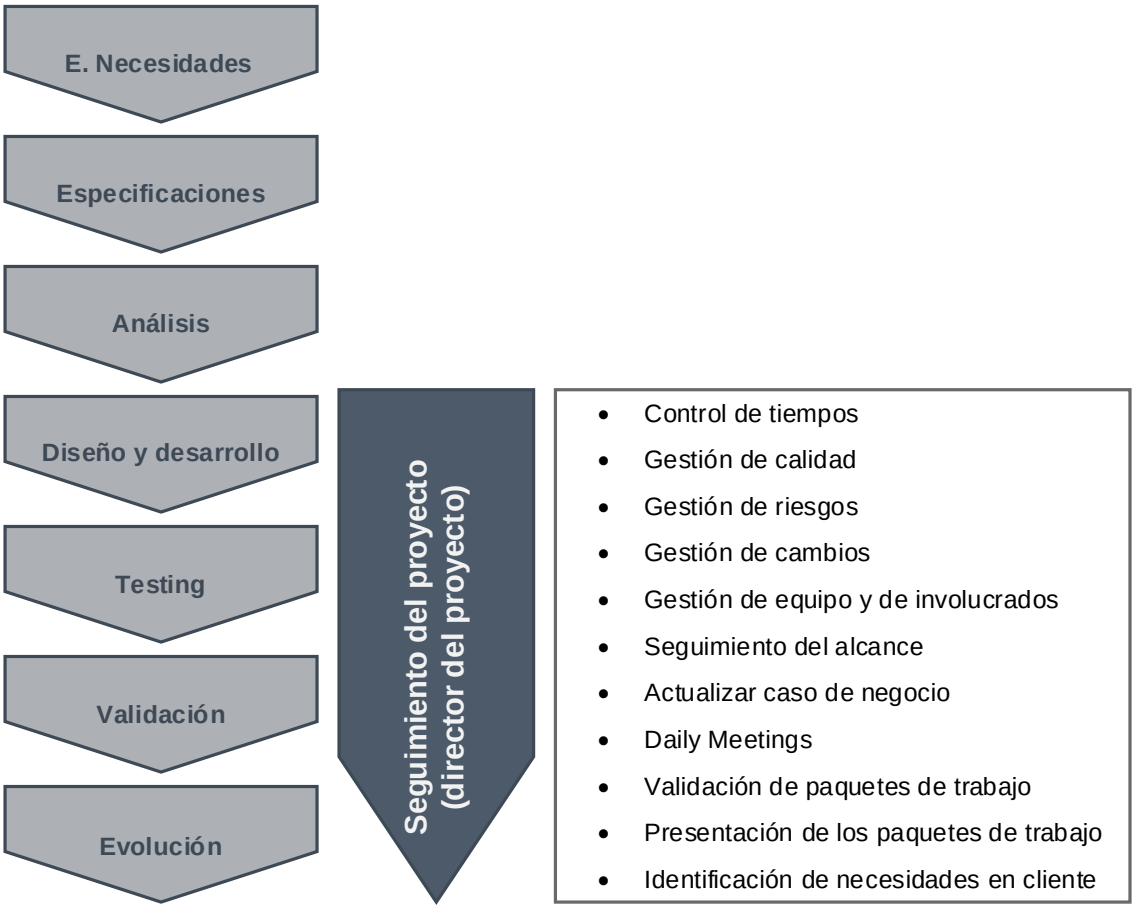
En el caso de que ya esté terminado, se debe hacer una reunión entre la junta del proyecto, y acordar el cierre del proyecto. Esta reunión contará con un acta donde se haga costar que todos los trabajos están terminados, y que tanto el cliente como la empresa de outsourcing están de acuerdo en cerrar el proyecto.

En dicha reunión se puede hacer constar la conveniencia de abordar una segunda fase o una evolución del producto, si se ve necesario, en todo caso si es importante que por lo

menos se deje constancia de las posibles mejoras que se podrían dar al producto entregado intentado abrir nuevas oportunidades de negocio con el cliente.

Estos proyectos cuentan con una garantía de unos meses, en los que se deben resolver posibles incidencias que surjan durante su funcionamiento. Esta garantía debe quedar bien definida, para que ambas partes tengan claro qué tipo de incidencias se cubren y que siempre sean de tipo bug, falla en el funcionamiento, un requerimiento mal entendido, por parte de la empresa de outsourcing a pesar de estar bien contemplado en la especificación, etc. Pero nunca una modificación en las especificaciones, esto no lo debería de cubrir la garantía del proyecto, porque en ese caso sería difícil dejar cerrado un proyecto, ya que estos siempre son susceptibles de cambios y mejoras.

4.9 **Seguimiento del proyecto**



Lo ideal es que el jefe de proyecto no participe del diseño y desarrollo en lo que se refiere a la codificación, sólo asesore al equipo, resuelva dudas, mantenga comunicación activa con el cliente y con la dirección de la empresa.

Además, es probable que el director de proyecto esté gestionando varios proyectos a la vez, por lo que en estas fases sus labores deberían ser:

4.9.1 Control de tiempos

Llevar un control de los tiempos con herramientas tipo Microsoft Project, Jira, etc. Esto permite al jefe de proyecto ser muy consciente del estado del proyecto, y prever retrasos o sobrecostes para poder anticiparse y tomar medidas correctivas.

Además, el uso de estas herramientas permite generar informes de forma ágil que pueden ser útiles para informar a la junta del proyecto, por ello se recomienda estar familiarizado

con estas herramientas y tener formación sobre ellas. El tiempo perdido en esta formación, es fácilmente rentabilizado, ya que son herramientas con multitud de funcionalidades y el ser capaz de usarlas ágilmente es un punto muy favorable. Al igual que con el resto de herramientas habituales como las de Office, el Visio, etc. Para mejorar la productividad es esencial una buena formación en estas herramientas ya que son de uso muy habitual y con buenas prácticas el ahorro de tiempo es apreciable.

4.9.2 Gestión de calidad

Verificar que se está cumpliendo con la calidad, al dividir el proyecto en historias de usuario, es fácil perder la perspectiva, es labor del director de proyecto que tiene una visión global del producto, comprobar que no se está excediendo en las funcionalidades ni por supuesto quedándose cortos, aunque es más habitual lo primero, ya que el equipo tiende a desarrollar nuevas funcionalidades o dar más prestaciones de las acordadas con el cliente, lo que es un problema y merma la calidad del producto.

4.9.3 Gestión de riesgos

Actualizar el registro de riesgos. Es muy importante hacer una gestión constante de los riesgos y mantenerlos actualizados e identificados.

4.9.4 Gestión de cambios

Gestionar incidencias, realizar informes o elevarlas si es necesario. Es importante que se mantenga firme con los stakeholders evitando cambios perjudiciales para el proyecto, que rompan acuerdos previos o que interrumpan el curso del sprint, siempre que sea posible.

Esta parte es la más complicada de cara a la gestión de los stakeholders porque es donde suelen estar las discusiones y los roces. El objetivo del director de proyecto debe ser:

- Salvaguardar la calidad y el alcance del proyecto.

  Se puede solicitar un cambio, que este fuera de los limites acordados en un principio para el alcance. En ese caso lo mejor es intentar convencer al cliente para que ese cambio se deje para una segunda fase del proyecto, o bien si no cede, habrá que modificar el alcance y por ende, la gestión de tiempos, costes y riesgos. Dejando constancia en un acta validada por la junta del proyecto que sirva como documento contractual.

- Salvaguardar la imagen de la empresa, impidiendo cambios de baja calidad que pongan en entredicho el buen hacer de la compañía y del equipo de desarrollo.

  Es habitual, que por necesidad de acortar tiempos, o reparar algún error, se pida algún cambio "*chapuza*", que no cumpla con las normas de calidad y estilo de la compañía. Una excepción para salir del paso. Es aconsejable no ceder en estas situaciones e intentar acordar con el cliente el cambio, pero de forma planificada, con estilo y de calidad. En el futuro, aunque quede constancia de que ese cambio de baja calidad fue una solicitud expresa del cliente, cuando este desarrollo provoque fallas o en el caso de una interface de usuario, se vea desastrosa, el responsable último siempre va a ser la compañía que firmó ese desarrollo. Esto provoca daños en la imagen de la empresa que no son cuantificables ni fácilmente valorables, pero sin embargo, van a estar ahí.

- Satisfacer completamente las necesidades del cliente.

  Al final el director debe mantener todo lo satisfecho que pueda al cliente, y cumplir con sus necesidades de ello va a depender buena parte de los proyectos futuros.

- Gestionar cambios de estructura que afecten a partes ya desarrolladas y que puedan provocar fallos futuros, problemas de mantenimiento o retrasos.

  Estos cambios son fruto de una mala planificación en las etapas tempranas, situaciones que no se contemplaron o que se contemplaron mal. Son siempre muy costosas para la empresa de outsourcing y para el propio cliente, y se debe poner todos los medios para evitarlas, de ahí la importancia de las primeras fases de planificación. Pero si al final se da, habrá que replantear el diseño inicial, reunir al equipo y entre todos hacer un replanteamiento que afecte lo menos posible al producto que está ya en producción.

  Para que estos cambios se puedan hacer de una forma ordenada y con un coste menor, es vital la calidad del código y de la arquitectura de la aplicación, si se ha previsto una arquitectura flexible, muy abstracta, con módulos lo más independientes posible, con pruebas unitarias para el testeo, etc. Estos cambios se harán de una forma mucho menos costosa. Si esto no ha sido así, entonces igual se hay que replantear, dependiendo del cambio, el empezar de nuevo.

- Impedir caer en sobrecostes que provoquen la falta de rentabilidad del proyecto.

  A la hora de gestionar los cambios, se debe informar a la junta del proyecto de forma adecuada, para que puedan acordar los costes de dichos cambios y que sean conscientes en todo momento del estado del proyecto para salvaguardar la rentabilidad del proyecto.

4.9.5 Gestión de equipo

Estar pendiente del progreso de los integrantes del grupo y motivarlos cuando sea necesario o emprender acciones. Durante el desarrollo se pueden dar disputas entre los integrantes del equipo que habrá que saber gestionar, además, algunas partes del proyecto pueden ser poco motivantes para algún miembro del equipo. Se debe intentar tener un registro del perfil de cada desarrollador, con sus puntos fuertes y sus puntos débiles, de forma que a la hora de asignar tareas, al mismo tiempo que se busca formar equipos multidisciplinares de forma que todos sepan de todo, también intentar que cada uno desarrolle más intensamente aquellas partes en las que se siente más cómodo.

Hay que intentar anticiparse a los conflictos o al menos, tenerlos previstos para cuando estos se producen poder mitigarlos eficazmente.

Asistir a los Daily Meetings para estar al tanto del progreso del proyecto e intentar facilitar recursos y ayudar de esta manera a resolver problemas que se puedan dar durante el desarrollo. Esto es una práctica esencial, para poder estar al tanto de los detalles del proyecto y poder ayudar si es necesario.

4.9.6 Gestión de involucrados

Uno de los riesgos, es la gestión de todos los involucrados del proyecto o stakeholders. Ni en Prince 2 ni en Scrum existe un proceso para gestión de involucrados como tal, así que, este autor lo introduce en el apartado de gestión de riesgos ya que como se explica a continuación es un riesgo que necesita ser controlado y planificado para lograr el éxito del proyecto.

Los involucrados del proyecto son cualquier persona o grupo de personas que puede influir en el proyecto o que son afectados por este (Pérez A. , 2015). En este sentido, es muy importante identificar a todos ellos, incluyendo los siguientes elementos:

- Nivel de poder y/o influencia.
- Grado de soporte hacia el proyecto.
- Riesgo de impacto al proyecto.

Es labor del jefe de proyecto intentar que los involucrados se vayan situando en posiciones favorables al proyecto, sobre todo si su grado poder es alto.

Se deben clasificar en función de su interés e influencia.

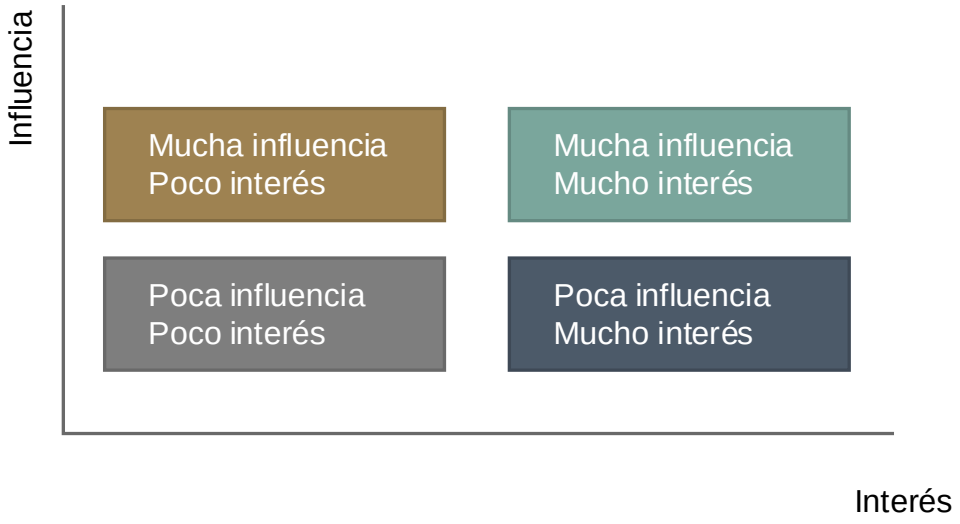


Ilustración 4.7. Clasificación stakeholders (autor)

Los pasos a seguir son:

1. Identificar a los interesados. Documentar la información relevante en cuanto a sus intereses, como están involucrados en el proyecto y su poder de influencia
2. Planificar la gestión de los interesados.
3. Gestionar su participación. Comunicarse con los stakeholders y trabajar en la consecución de sus necesidades de acuerdo con un plan definido y gestionar sus expectativas para aumentar la posibilidad de aceptación del proyecto y anticiparse a futuros problemas.
4. Control y seguimiento. Comunicación constante para comprobar su estado y si se están cumpliendo sus expectativas y en caso de que no sea así poder tomar acciones a este respecto.

En función del mayor poder que tenga un interesado u otro, puede variar el alcance del proyecto, por ejemplo si el involucrado de mayor peso es del departamento de contabilidad, es probable que se necesite que el proyecto se haga en el menor tiempo posible, penalizando la calidad para abaratar costes. Aunque esto en el sector industrial, no es habitual.

Es una buena práctica que estas estrategias de gestión y comunicación sirvan para involucrar a los interesados en las principales decisiones del proyecto y así facilitar su ejecución. (Lledó, 2017)

4.9.7 Seguimiento del alcance del proyecto

Poniendo limites en las fases del proyecto si es necesario, es importante hacer un buen seguimiento del proyecto de forma que no se exceda el alcance y no se incurran en sobrecostes para la empresa.

Validar los paquetes de trabajo ya dados por terminados por parte del equipo para comprobar que cumplen con las especificaciones acordadas con el cliente.

Presentar los paquetes de trabajo terminados al cliente. Es una parte fundamental ya que no sirve de nada hacer un buen trabajo si no se explica correctamente al cliente, para que este se involucre, sea consciente de este trabajo, de las dificultades que con llevo y de cómo se han solventado. Al cliente se le debe explicar bien cuál es la calidad del producto entregado para que valore el trabajo realizado por el equipo.

4.9.8 Actualizar el caso de negocio

Alertando en el caso de que se detecte de que el producto deja de tener sentido para el cliente o para la empresa. Es una constante, vital en la metodología Prince2, verificar que el proyecto sigue teniendo sentido. Esta parte cobra mucha importancia en proyectos de larga duración, donde puede ocurrir que otra tecnología, un cambio de estrategia en el cliente, etc., haga que el proyecto ya no sea necesario. Es importante evaluar esta situación para poder anticiparse.

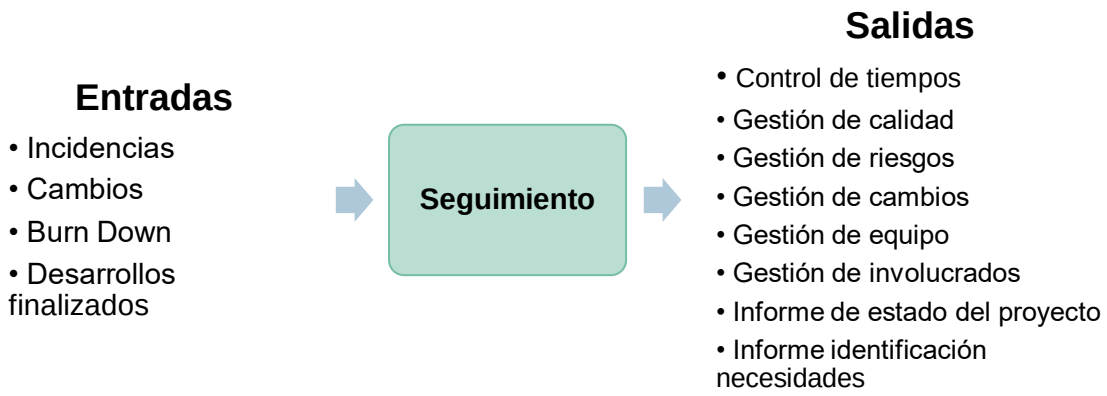
4.9.9 Gestión documental

Se debe tener una gestión documental adecuada, que facilite las búsquedas de ciertos documentos en un momento dado, que no incurra en palabrería, la documentación ha de ser de fácil lectura y compresión, una documentación ágil.

La gestión de lecciones aprendidas aportará mucho valor al equipo proveyendo de agilidad a este en futuros proyectos.

4.9.10 Identificar necesidades en el cliente

Con cada proyecto surgen oportunidades para nuevos proyectos. El director del proyecto es la persona que conoce las necesidades del cliente y tiene que estar atento a nuevas ideas de mejora en el negocio del cliente, para cuando se dé la circunstancia poder hacer propuestas de valor.



5 Validación de la metodología hibrida

Finalmente se expone un ejemplo de gestión de un proyecto real de software industrial, y se describe como se abordó cada una de las fases a partir de lo estudiado en los apartados anteriores.

Se gestionó un pequeño proyecto, de aproximadamente un mes de duración, siguiendo la metodología vista en el apartado anterior. Se ira viendo fase a fase como se hizo el seguimiento de dicho proyecto. Se focalizará en la gestión del proyecto y no en la parte técnica.

Los nombres de las empresas, los involucrados y otros datos que carecen de interés para la gestión que nos ocupa, son ficticios para salvaguardar su privacidad.

5.1 **Contexto**

Participantes:

- Ipsum, S.L., empresa dedicada al desarrollo de software outsourcing.
- Tornillería Emilio, fábrica de tornillos.

Tornillería Emilio, decide instalar un horno donde poder calentar piezas de acero, para después crear el alambre que posteriormente convertirá en tornillos.

Una de las fases de este proyecto consiste en realizar una pantalla para los horneros donde puedan visualizar el estado del horno y de esa manera poder influir en el calentamiento de las piezas para lograr un resultado óptimo de temperatura a la hora de la extracción de las mismas para su posterior laminación.

Ipsum son especialistas en desarrollo software de bajo coste, una pyme de 50 trabajadores cuyo modelo de negocio es el de '*software factory*', con varios jefes de proyecto experimentados en este sector y un equipo de desarrolladores muy jóvenes y con un alto grado de rotación. Su localización está en la misma localidad que la tornillería.

5.2 **Expresión de necesidades**

Se convocó a IpsumCoop, a una reunión en el cliente, en esa reunión, lo ideal según lo visto en este documento en que asistiesen:

- El responsable del proyecto en la tornillería, quién tomará las decisiones en el cliente.
- El responsable en Ipsum, que deberá percibir la necesidad para luego dirigir el proyecto en la empresa de outsourcing para cubrir totalmente la necesidad especificada
- Un maestro hornero, que aclarará las dudas sobre las necesidades reales que tiene y que funcionalidades serían las que más valor le aportarían.
- Un responsable del sistema eléctrico, encargado de la parte del horno y que resolverá las dudas sobre la viabilidad de ciertas funcionalidades de las que se necesitará la información gestionada por los PLCs de los cuáles es responsable.

En un anterior proyecto, en el que no se había realizado esta metodología y no había asistido ni el maestro hornero ni un representante de los eléctricos se había incurrido en retrasos por el siguiente motivo:

En el caso del maestro hornero, al definir la interfaz se había definido con unos colores que representaban en RGB el color del acero a ese grado de temperatura, estos colores resultaban muy chillones al ser rojos fuertes y amarillos muy claros, al llegar estas pantallas a producción, se habían producido quejas de los operadores por ser pantallas con colores tan fuertes que les cansaba la vista al tener que estar visualizándolas ocho horas de continuo. Se tuvieron que modificar las pantallas para adaptarlas a los operadores.

Para el caso de los eléctricos, que son los responsables de la programación de los PLC’s, la empresa de outsourcing para hacer un desarrollo más rápido y fiable había solicitado duplicar las comunicaciones de estos PLC’s para poder trabajar con ellas en desarrollo sin que afectase a producción. El responsable en el cliente aseguró que esto no era viable, por lo que la empresa externa tuvo que desarrollar también un simulador de estas señales lo que le llevo un desarrollo más extenso y menos fiable.

Estos dos retrasos se solucionaron esta vez al asistir el maestro hornero y validar la especificación de interfaz de usuario, y el eléctrico al corroborar que sí que se podía duplicar dicha comunicación lo que agilizo el desarrollo.

Los documentos que generó el jefe de proyecto:

5.2.1 Acta de la reunión

Cliente	Tornillería Emilio, S.A.
Proyecto	Visualización horno
Fecha	01/06/2019
Lugar	Tornillería Emilio

Asistente	**Empresa**	**Cargo**
Francisco Hermida	*Tornillería Emilio*	*Responsable IT*
Manuel Suárez	*Ipsum*	*Jefe de proyecto*
Iván García	*Tornillería Emilio*	*Maestro hornero*
Luis Menéndez	*Tornillería Emilio*	*Jefe Mantenimiento eléctrico*

Objeto

Diseño de la visualización del horno para el pulpito de los horneros. Que sirva para comprobar el estado del horno y llevar un control de temperaturas sobre él.

Resumen

<En esta parte se recopiló lo hablado en la reunión, intentando ser fiel y objetivo a todo lo comentado por cada uno de los asistentes.
Se hicieron las preguntas adecuadas que consiguieron extraer la mayor información posible sobre la necesidad del cliente, y ayudaron a definir bien el propósito del proyecto, pero sin indicarle lo que debería hacer y cómo, que suele ser un error habitual en los consultores>

Reparto de responsabilidades y tareas

Tarea	**Responsable**	**Fecha requerido**
Realización de una especificación y una maqueta a partir de lo hablado	*Jefe proyecto Ipsum*	*10/06/2019*
Estudio de las señales necesarias para exportar los datos para la visualización	*Jefe Mto. eléctrico*	*10/06/2019*

Fecha próxima reunión	**10/06/2019**

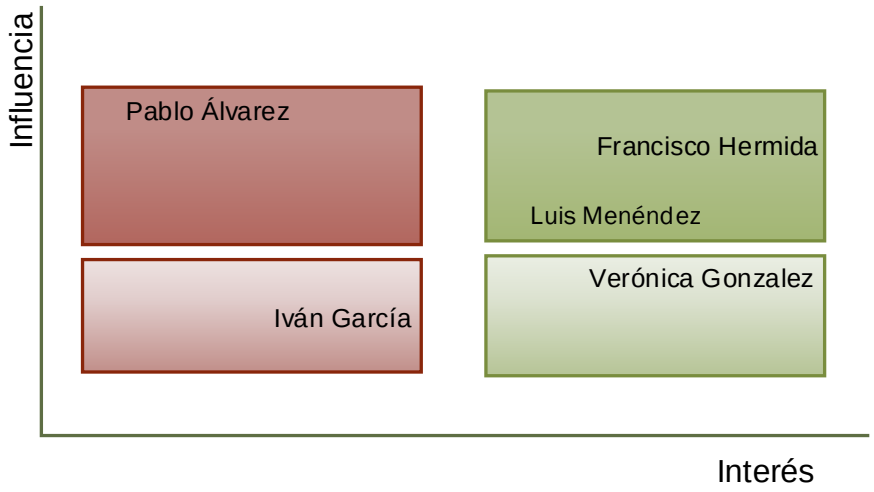
5.2.2 Registro de stakeholders

En la reunión inicial se estuvo atento a recopilar datos sobre los involucrados. Se les preguntó también si usaban herramientas de comunicación tipo "*Teams*" o "*Skype*", con el objetivo de mantener una relación fluida. Se usó Microsoft Teams con el representante en cliente.

Cliente	Tornillería Emilio, S.A.
Proyecto	Visualización horno
Fecha	01/06/2019

Nombre	Empresa	Cargo	Comentario	Contacto
Francisco Hermida	*Tornillería Emilio*	*Responsable del proyecto en IT*	*Muy trabajador y competente, está muy interesado en que este proyecto salga para adelante.* *Persona práctica que le gusta lo sencillo.*	*Francisco.hermida@tornilleria.es* *620285367*
Iván García	*Tornillería Emilio*	*Maestro hornero*	*Persona seria, con cierta inquietud a que la nueva visualización lo haga prescindible.* *Tiene claro lo que necesita para hacer su trabajo*	
Luis Menéndez	*Tornillería Emilio*	*Jefe Mto. Eléctrico*	*Está desbordado de trabajo, con muchos proyectos abiertos. Es difícil que atienda a los emails, y se retrasa mucho en las peticiones.*	*Luis.Menéndez@tornilleria.es* *620286980*
Verónica González	*Ipsum*	*Directiva*	*Este proyecto le parece importante para la empresa porque puede abrir negocio en cliente.* *No dudará en aportar recursos si es necesario*	*Veronica.gonzalez@ipsum.com* *620505120*
Pablo Álvarez	*Tornillería Emilio*	*Responsable IT*	*No le da demasiada importancia al proyecto, no está en los objetivos de este año, y considera que no le aporta valor a él ni a la fábrica*	*Pablo.Alvarez@Tornilleria.es* *620414250*

5.2.3 Matriz involucrados



5.2.4 Informe preliminar del proyecto

Al tratarse de un proyecto de poca envergadura, se envía un correo electrónico al responsable del departamento en Ipsum, indicando:

- El alcance del proyecto de forma superficial
- Análisis de riesgos detectados, sobre todo incidiendo en el poco interés de alguno de los involucrados que pudiese echar por tierra el proyecto o anteponer otros proyectos a este.
- Estimación inicial de tiempo
- Los recursos necesarios para llevarlo a cabo.

5.2.5 Gestión documental

Mientras se obtiene el visto bueno de dirección y se asignan los recursos del proyecto, se crea la estructura para la gestión documental y se va registrando en ella los documentos actuales.

El estándar de nomenclatura que se emplea es:

NombreEmpresa.Proyecto.Fecha.Asunto

Para el acta por ejemplo: *TE.VH.01062019.Acta inicial.docx*

Para este ejemplo, se usa *Confluence*, que es una herramienta de software que permite hacer esta gestión, en ella se creó el registro de los documentos con un enlace hacia ellos.

Titulo Documento	Resumen	Autor	Fecha	Enlace
NombreEmpresa.Proyecto.Acta_20190402	Acta de inicio de proyecto. Se define las necesidades del cliente	Jose Alvarez	02/04/2019	\\servidor\Documentacion\Cliente\Proyecto
NombreEmpresa.Proyecto.Acta_20190508	Acta de especificaciones. Propuesta y recogida de historias de usuario	Jose Alvarez	05/08/2019	\\servidor\Documentacion\Cliente\Proyecto

Ilustración 5.1. Registro de documentos (autor)

Los documentos se introdujeron en una carpeta compartida en el *Drive de Google*, en función del tipo de documento, se dieron permisos a los diferentes involucrados.

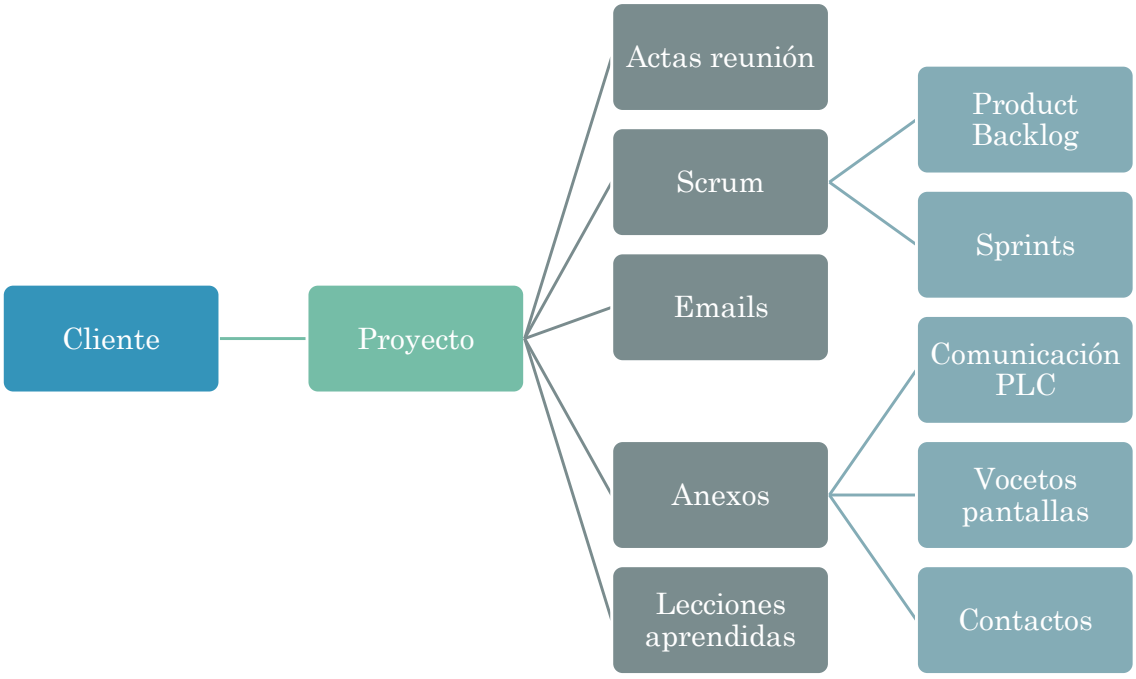


Ilustración 5.2. Ejemplo de estructura de directorios (autor)

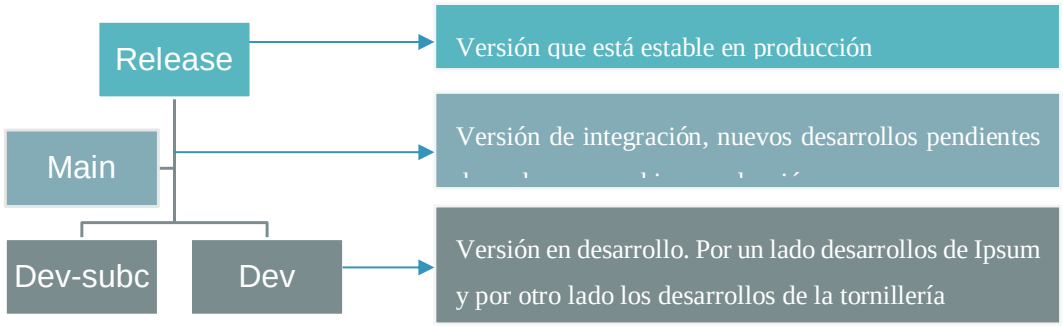
Lo ideal es contar con herramientas tipo *Confluence* o *Jira*, que están desarrolladas para llevar este control documental, pero de no existir, se puede crear algo similar simplemente creando un documento como una hoja *Excel*, que sirva de guía de todos los documentos almacenados, de forma que se puedan buscar un documento de forma ágil.

5.2.6 Plan de calidad

Si se van a usar herramientas tipo *Sonar Lint*, es importante dejar constancia de ello y hacer una propuesta sobre las reglas y el nivel de aceptabilidad para el código. En este caso no se usó ningún tipo de control de calidad sobre el código más allá de la supervisión del propio jefe de proyecto en la empresa externa.

En este ejemplo se usará *Microsoft Team Foundation Server* como herramienta de control de código fuente en el servidor que tiene el propio cliente, y que creó una rama donde introducir el nuevo código.

La estrategia de ramas quedó de la siguiente forma:



5.2.7 Conformación del equipo

Una vez que se tuvo el visto bueno del responsable del departamento, se conformó el equipo, que este ejemplo estaba formado por cuatro personas:

- Yonatan Álvarez, 4 años de experiencia en tecnologías .Net.
- Alfonso Menéndez, 2 años de experiencia en tecnologías .Net
- Adrián Gómez, 1 año de experiencia en bases de datos y .Net
- Manuel Suarez, el jefe de proyecto, con más de 15 años de experiencia en el sector.

Después se mantuvo una reunión con el equipo donde:

- Se explica el proyecto.
- Se resuelven dudas y de no poder ser resueltas, se toma nota para preguntar al cliente.
- Se comunica donde estará almacenada la documentación y de qué manera deberá ser registrada.

- Se definen los estilos de desarrollo de código y de interfaz de usuario.
- Se comunica el plan de calidad y de testeo de código.
- Se convoca para una segunda reunión donde los miembros del equipo tendrán que aportar ideas de cómo llevarlo a cabo y se debatirá sobre la maqueta que se propondrá al cliente.

En otros proyectos se gestionó al equipo de forma más autocrática, dándoles simplemente la especificación de lo que tenían que hacer y cómo. El resultado fue que los desarrolladores estaban menos motivados con su trabajo por lo que fueron más lentos y con menos calidad, aun así el trabajo se culminó antes, porque se dedicaron a hacer lo que se les pidió sin aportar valor al proyecto.

El resultado en este caso fue un proyecto más rápido, menos fiable y más pobre.

Sin embargo, como resultado de la gestión actual de equipo, más participativa y fomentando la participación, se logra implicación de la mayoría de los miembros del equipo, más interacción y colaboración entre ellos por lo que se estrechan sus relaciones. Más creatividad, e ideas que se trasladan al cliente y muchas de ellas son aceptadas y aportan valor al cliente, estas nuevas ideas, culminan en nuevas especificaciones que alargan la estimación inicial pero que cuentan con el apoyo del cliente al que generan valor y marcan la diferencia entre trabajar con esta empresa de outsourcing u otra que simplemente realice lo que se le pide.

5.3 **Especificaciones**

Ya se ha mantenido una segunda reunión con el equipo, mediante la cual se ha consensuado el proyecto que somos capaces a realizar para cubrir las necesidades del equipo.

Se realizan los esquemas con los paquetes de trabajo que se ven necesarios a priori y una maqueta de la interfaz de usuario.

5.3.1 Paquetes de trabajo

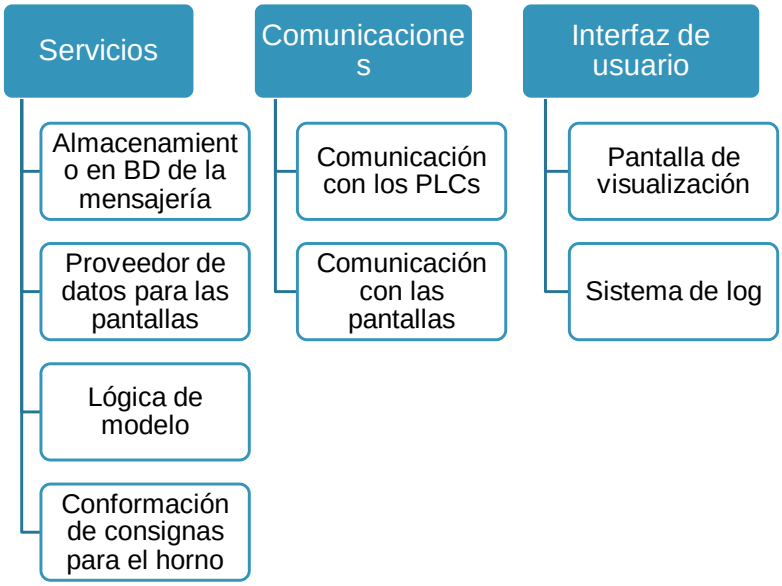


Ilustración 5.3. Paquetes de trabajo (autor)

5.3.2 Esquema de arquitectura

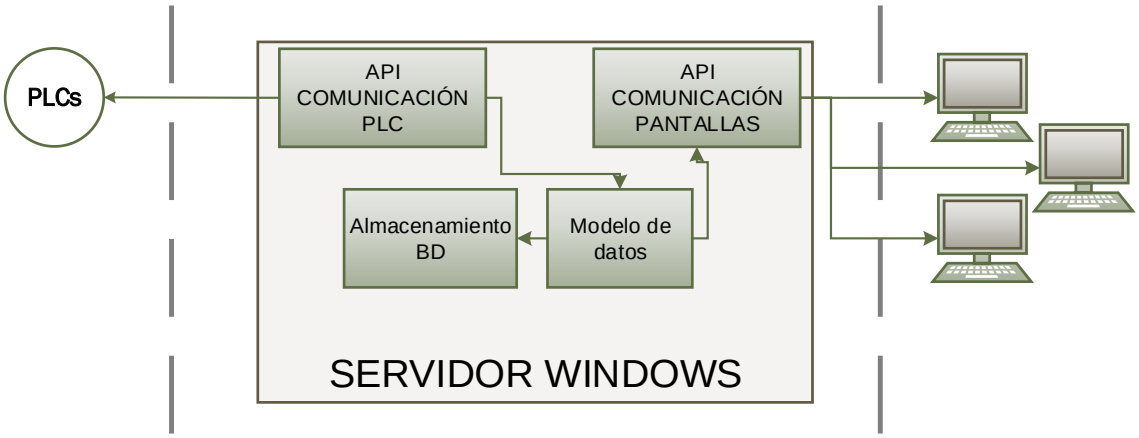


Ilustración 5.4. Arquitectura proyecto (autor)

5.3.3 Interfaz de usuario

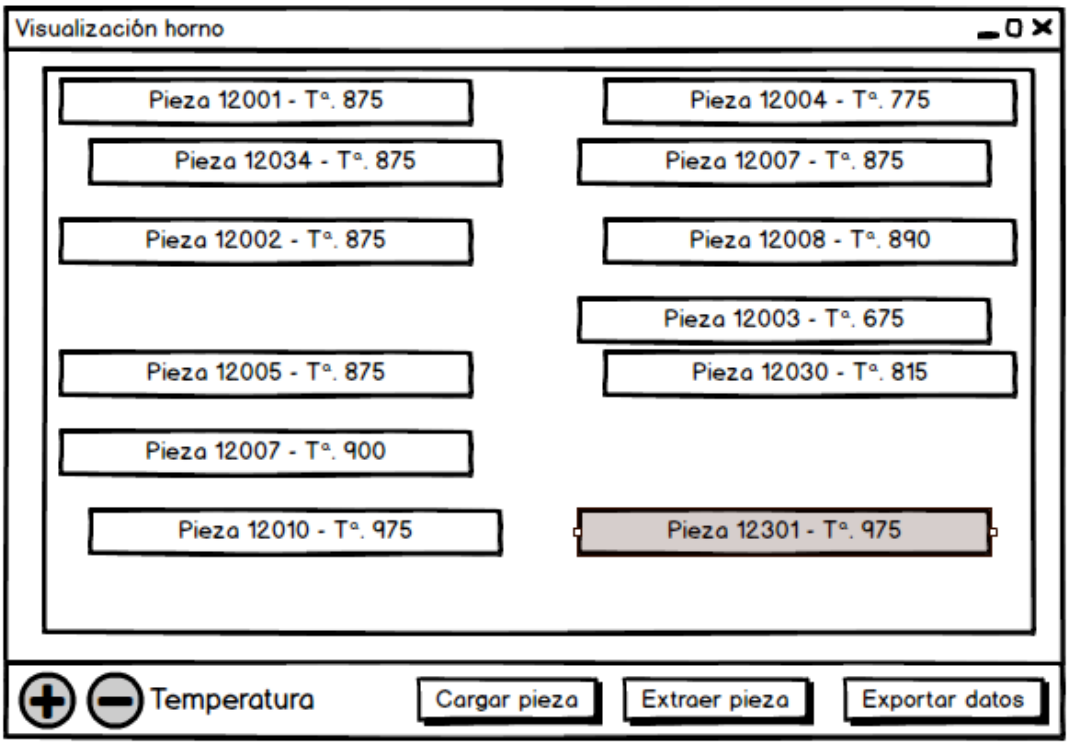


Ilustración 5.5. Boceto inicial de pantalla (autor)

5.3.4 Toma de requisitos

Se propone otra reunión con el cliente, donde se explican los puntos anteriores, se busca la discusión para analizar junto con los implicados si la propuesta realizada por el departamento de desarrollo cumple las expectativas y cubre correctamente las necesidades.

Cliente	**Tornillería Emilio**
Proyecto	**Visualización hornos**

Fecha	**10/06/2019**
Director de proyecto	Manuel Suárez

ID	Titulo	Tipo	Descripción	Motivo	Complejidad	Impacto
VH01	*Comunicación Nivel 1*	*Comunicación*	*Se debe crear una comunicación con el PLC y mapear los mensajes necesarios para obtener toda la información requerida, con un protocolo que asegure la recuperación en caso de pérdida de conexión.*	*Para los movimientos del horno, el feedback de la combustión y el envío de nuevas consignas, es necesario establecer esta conexión.*	*Alto*	*Alto*
VH02	*Almacenamiento en BD*	*Persistencia datos*	*Se deben almacenar todos los datos de la mensajería en la base de datos, teniendo que crear el diagrama de base de datos, que será un modelo relacional que asegure la robustez de los mismos*	*Registro del estado del horno. Es necesario garantizar estos datos por lo menos durante 10 años*	*Baja*	*Alto*
V03	*Modelo de datos*	*Lógica*	*Se debe desarrollar un modelo capaz de actuar sobre el horno y dar las consignas para el calentamiento de las piezas adecuado para conseguir una extracción de piezas en la temperatura objetivo*	*Automatizar el trabajo del hornero*	*Alta*	*Medio*

(Ejemplo de toma de requisitos de alto nivel, no hará falta entrar en el detalle, se trata de las funcionalidades básicas)

5.3.5 Acta final

Un punto importante de esa reunión fue analizar los riesgos que existían en ese proyecto y como mitigarlos.

Por ejemplo:

Se pidió a Mto. Eléctrico que doblasen las comunicaciones para que se pudiesen recibir esas comunicaciones también en el servidor de desarrollo y de esa manera poder trabajar con datos reales y ser más fiables a la hora de subir a producción.

Era probable que Mto. Eléctrico no pudiese hacerlo porque sus PLCs estuviesen muy sobrecargados, porque no sepan hacerlo o simplemente porque no les aporta valor directamente a ellos y prefieren no hacerlo.

En ese caso hay que valorar un plan B para mitigar este problema, este plan B era crear unos procesos a modo de Gateway que reenviasen la comunicación del servidor de producción al de desarrollo tal y como llega inicialmente a este.

Tras finalizar esta reunión se reenvió un acta al cliente como en la del punto anterior, pero esta vez indicando con más precisión todos los detalles del proyecto, es un acta a modo de contrato, donde ya se dejó constancia de forma clara cuales son los puntos que se abordaban en el proyecto y cuáles no, adjuntando la toma de requisitos inicial que es una toma de requisitos de alto nivel ya que a la hora del desarrollo es cuando se abordó todo con mayor detalle.

En este acta también se hizo una estimación de los tiempos para que tanto cliente como desarrolladores tuviesen un objetivo claro y se pusiesen a trabajar.

5.4 **Análisis**

5.4.1 Pila de Backlog

Se convoca a una reunión donde asistió el cliente y el equipo de desarrollo, se estudió la fase a analizar, se analizó la situación y se conformaron las historias de usuario.

Esta reunión duró unas 6 horas. Es conveniente contar al menos con una sala que tenga pizarra, y si puede ser proyector o similar, ya que son herramientas que ayudan a analizar los problemas y que todos salgan con las ideas claras de lo que habrá que hacer en esta fase.

Ejemplo de historia de usuario, en el proyecto que estamos trabajando:

Cliente	Tornillería Emilio
Proyecto	Visualización Horno
Fecha	11/06/2019
Responsables	Manuel Suárez
	Francisco Hermida
	Adrián Gómez
Objeto	Mapeo del Mensaje de combustión del horno

Descripción
El PLC del horno tiene los datos de combustión del horno. Se trata de crear un mensaje, para poder solicitar estos datos al plc, y recibirlos desde el servidor Windows. *El mensaje consta de 128 bytes, donde todos los datos están en formato carácter, así:* *0-15 bytes. cabecera del mensaje según el formato FVH-2100* *15-128 bytes. Datos de combustión:* *15-24 bytes. Fecha en formato YYYYMMDD* *24-30 bytes. Hora en formato HHMMSS* *...* *Una vez recibido el mensaje en el servidor Windows, se deberá convertir el array de bytes en un objeto que cumpla con la interfaz IMensajeComustion, y cada uno de los datos convertido a su tipo de dato correspondiente.* *Además se almacenará en fichero en la carpeta MensajesCombustion, donde quedarán almacenados por día, los mensajes recibidos en el último mes. Con el primer mensaje del día se borrarán todos los mensajes correspondientes al mismo día del mes anterior.*

Validación
Se crearán los test unitarios necesarios para validar que el mensaje está bien mapeado, para ello se debe de crear un mensaje de pruebas, conocido, que se convertirá a array de bytes, y tras pasar por la comunicación, se comprobará que al convertirlo en el IMensajeCombustion se mapean bien los datos campo a campo. *Además se debe de comprobar que pasa cuando los datos llegan mal y mitigar este problema sin que afecte al resto del sistema y dejando un log claro donde se evidencie el incidente.*

Prioridad	Alta	**Tiempo estimado**	12 horas

Una vez conformadas todas las horas de usuario, el cliente las priorizó y a partir de aquí se comienza el debate sobre la preparación del primer sprint.

5.4.2 Primer Sprint

Se acuerda con el cliente, que los Sprints serán de dos semanas de duración. Considerando trabajo efectivo de 6 horas diarias (eso a pesar que la jornada laboral es de 8.65 horas, pero se considera y se estima sobre 6 horas como unidad efectiva de trabajo).

El cliente va explicando las historias de más prioridad y el equipo una vez que las entiende estima las horas que le llevará darlas por terminadas, de esa manera queda configurado el primer sprint de esta forma:

Historia	Tarea	Asignada	Esfuerzo (estimado)	Esfuerzo (restante)	M	X	J	V	L	M	X	J	V	L
VHC01	Comunicación. Msg combustión	Adri	12	0										
VHC02	Comunicación. Msg validación	Adri	12	0										
VHC03	Comunicación. Msg Carga / Descarga	Adri	12	0										
VHC04	Comunicación. Msg Envio Consignas	Adri	12	0										
VHC05	Comunicación. Msg Mapa Horno	Adri	12	0										
VHBD01	API BD. Creación BD relacional. Esquema	Yoni	24	0										
VHBD02	API BD. Creación BD relacional. Tablas	Yoni	24	0										
VHBD03	API BD. Creación BD relacional. Procedimientos de mto.	Yoni	12	0										
VHI01	Interfaz cliente. Modelo MVP. Estructura de aplicación	Alfonso	6	0										
VHI02	Interfaz cliente. Modelo MVP. Vista de la pantalla	Alfonso	24	0										
VHI03	Interfaz cliente. Modelo MVP. Presenter de la pantalla	Alfonso	12	0										
VHI04	Interfaz cliente. Modelo MVP. Modelo de la pantalla	Alfonso	18	0										
	TOTAL		180	180	180	180	180	180	180	180	180	180	180	180

Ilustración 5.6. Configuración primer sprint (autor)

Podemos observar el gráfico de Burn Down del que se parte:

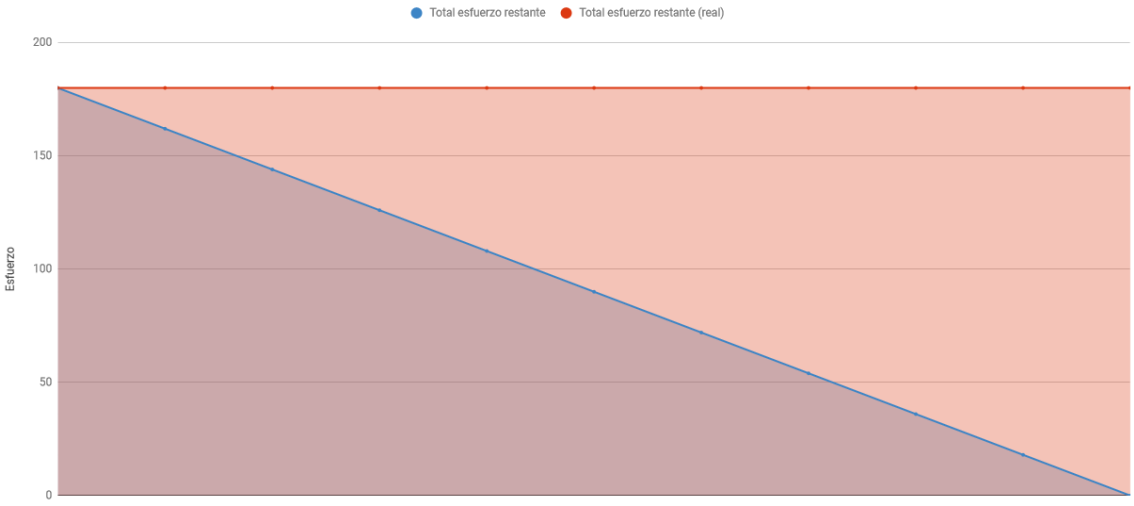


Ilustración 5.7. Gráfico Burn Down inicial (autor)

5.5 **Diseño y desarrollo**

El equipo comenzó a trabajar y la labor del jefe de equipo fue resolver dudas y proveer que no les faltase de nada para que pudiesen hacer su trabajo, resolviendo dudas incluso comunicándose con el cliente para ello. Y gestionando con dirección los materiales necesarios, etc.

Además llevó un seguimiento del proyecto para verificar que se estaba cumpliendo con el objetivo en cuanto a plazos y calidad.

Lo ideal, es una herramienta que en función del tiempo restante a la tarea que rellenen los desarrolladores, dibuje el gráfico, de no contar con dicha herramienta no es difícil programar una hoja Excel que cumpla esta función.

En este ejemplo, podemos observar a mitad del sprint, como está resultando.

Historia	Tarea	Asignada	Esfuerzo (estimado)	Esfuerzo (restante)	M	X	J	V	L	M	X	J	V	L
VHC01	Comunicación. Msg combustión	Adri	12	0	7	5	5	0						
VHC02	Comunicación. Msg validación	Adri	12	6	12	12	6	6						
VHC03	Comunicación. Msg Carga / Descarga	Adri	12	12	12	12	12	12						
VHC04	Comunicación. Msg Envio Consignas	Adri	12	12	12	12	12	12						
VHC05	Comunicación. Msg Mapa Horno	Adri	12	12	12	12	12	12						
VHBD01	API BD. Creación BD relacional. Esquema	Yoni	24	10	20	16	10	10						
VHBD02	API BD. Creación BD relacional. Tablas	Yoni	24	21	24	24	21	21						
VHBD03	API BD. Creación BD relacional. Procedimientos de mto.	Yoni	12	12	12	12	12	12						
VHI01	Interfaz cliente. Modelo MVP. Estructura de aplicación	Alfonso	6	0	0									
VHI02	Interfaz cliente. Modelo MVP. Vista de la pantalla	Alfonso	24	2	22	15	10	2						
VHI03	Interfaz cliente. Modelo MVP. Presenter de la pantalla	Alfonso	12	12	12	12	12	12						
VHI04	Interfaz cliente. Modelo MVP. Modelo de la pantalla	Alfonso	18	18	18	18	18	18						
	TOTAL		180	117	163	150	130	117	117	117	117	117	117	117

Ilustración 5.8. Seguimiento Sprint 1 (autor)

Analizando los datos podemos ver que ya hay dos tareas terminadas, pero necesitamos el análisis gráfico para poder ver de forma ágil si está habiendo problemas o por el contrario está saliendo un buen sprint.

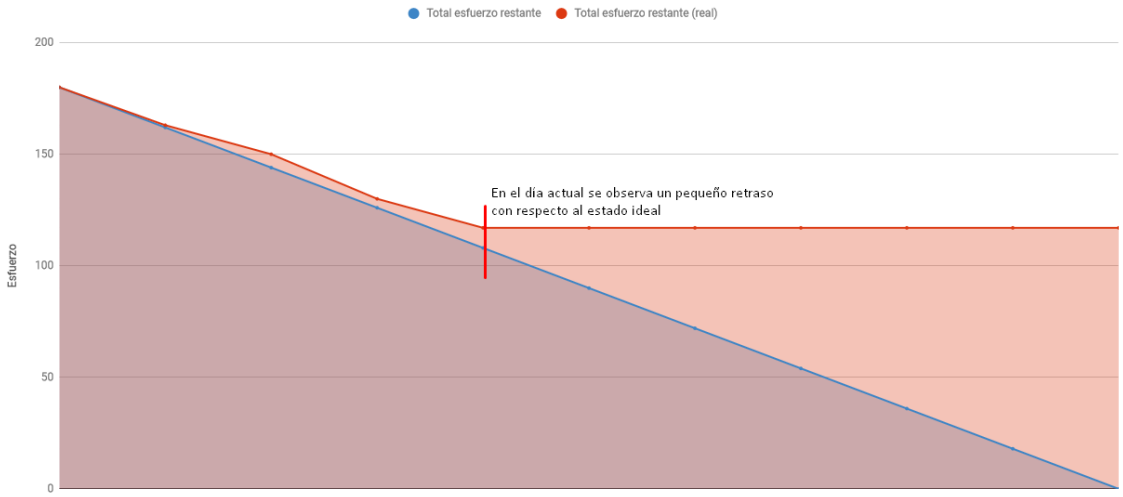


Ilustración 5.9. Gráfico Burn Down sprint 1 (autor)

En este gráfico vemos que aunque hay un pequeño retraso, estaba siendo un sprint muy bueno, ese retraso es más o menos normal y nos indica que es probable que lleguemos al final culminando prácticamente todas las tareas asignadas.

Sin embargo, si queremos entrar más en detalle, conviene analizar las gráficas individuales del equipo.

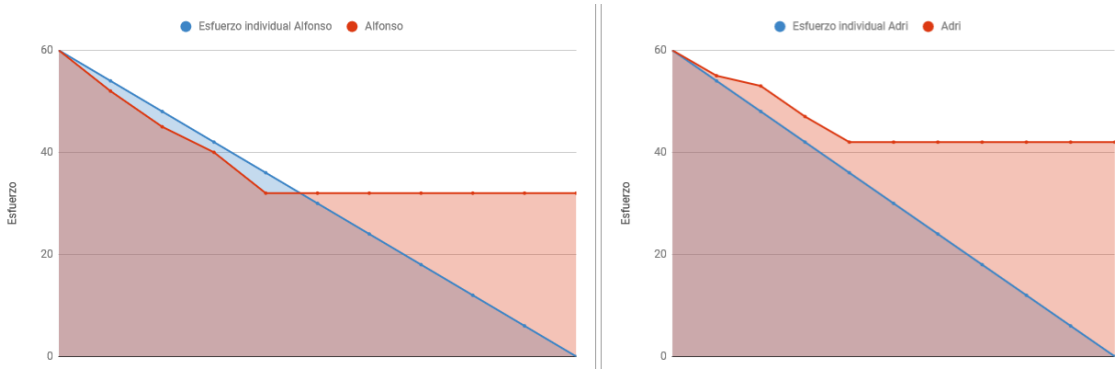


Ilustración 5.10. Gráficas Burn Down individuales (autor)

Ahora si se detecta problemas, se ve como Adrián iba con retraso, y además con tendencia a que se incremente, mientras que Alfonso iba por encima de lo estimado.

En este caso una medida que se adoptó es en función de los problemas que estaba teniendo Adrián, mirar en qué medida Alfonso que estaba teniendo más tiempo, lo podía ayudar y de esa manera llegar los dos bien al final de sprint.

Los problemas más habituales durante el sprint:

- Una tarea que se estimó mal porque era una tarea desconocida, en una tecnología nueva y al final va a llevar mucho más tiempo, o en el peor de los casos, no resuelve el problema para el que se había planteado.

  En este caso, lo ideal es que este riesgo hubiese estado contemplado y ahora se adoptase la medida para mitigarlo, aunque ello va a provocar inevitablemente retraso al proyecto.

- Una incidencia en el cliente, que requiere que se solvente urgentemente sin poder esperar al siguiente sprint. En muchos casos el propio jefe de proyecto que no participa directamente en el sprint puede intentar resolverla salvaguardando el resultado de sprint, pero si esto es muy habitual, se pueden reservar unas horas en la propia planificación del sprint para estos casos.

- Una baja de un compañero, bien por enfermedad, bien porque lo necesiten de urgencia en otro proyecto en el que estuviese anteriormente o porque la propia empresa necesita reunirse con él, y eso hace que vaya a tener menos tiempo

disponible del planificado. Al igual que en el caso anterior, el jefe de proyecto puede intentar mitigar este efecto y colaborar el mismo en las tareas que estaba desarrollando para ayudarlo a cumplir con el plazo.

- En épocas de muchos festivos, por ejemplo navidades, se puede aprovechar a que, aunque el sprint sea de dos semanas, si la siguiente semana sólo tiene dos días laborables, en vez de proponer un nuevo sprint, se puede alargar estas dos semanas, a modo de tareas de recuperación y aprovechar para estabilizar el proyecto, finalizando tareas que estaban un poco con hilos, o temas estructurales de documentación, etc.

Cuando estos problemas ya rompen completamente el sprint, es recomendable parar y organizar un nuevo sprint desde el inicio.

Para el seguimiento diario en este ejemplo se cuenta con la herramienta Reedbooth que virtualiza una pizarra de Kanban con cuatro pilas:

- Tareas de Sprint. El conjunto de todas las tareas, inicialmente en el sprint todas estarán en esta pila.
- Tareas en desarrollo, el desarrollador cuando comienza a desarrollar una tarea, la mueve de la pila de sprint, a esta otra pila, de esta forma todo el equipo visualiza que se está desarrollando ya esa tarea.
- Tareas en pruebas, una vez desarrolladas, se pasa a la fase de pruebas de integración, donde deberán permanecer un tiempo hasta que se dan por válidas y se cierran.
- Tareas atascadas, tareas que están pendientes de alguna cosa por parte del cliente, dependen de otra tarea que todavía no se terminó o simplemente se llegó a un obstáculo y mientras se busca una solución se decide seguir con otra tarea.

Este Kanban forma parte de la comunicación del equipo porque de un vistazo se puede observar con que tarea está cada uno, cuales ya están avanzadas y en fase de pruebas o por el contrario cuales están teniendo problemas.

Además, Reedbooth, permite comunicarse a través de la propia herramienta, por lo que se podrán asociar comentarios a las tareas, y a la hora de hacerles un seguimiento es fácil ver la trayectoria que tuvo esta tarea.

A parte de esto, diariamente se hacen reuniones del equipo de 15' de duración, donde cada miembro del equipo expone su situación y entre todos colaboran para ayudarse en las tareas del día a día.

5.6 **Debugging, testing y validación**

Forma parte del sprint, para validar cada historia de usuario, se pasaron las pruebas unitarias diseñadas para ello, y además un compañero o el propio jefe de proyecto, verificó que se cumplió con las especificaciones dadas para esa tarea y dejó constancia de ello a través de un checklist donde aparecen las funcionalidades y si cumple o no con ellas.

Cliente	Tornillería Emilio
Proyecto	Visualización Horno
Fecha	20/06/2019
Director proyecto	Manuel Suárez
Historia de usuario	VHC01

	Si	No	Comentario
Se recibe el mensaje del plc con todos los datos de combustión			*Falta el dato de la zona superior lado izquierdo*
Se mapea correctamente en un objeto IMsgCombustionHorno			
Se crea un fichero en el día correspondiente del mes en la estructura de ficheros acordada			
Se almacena el mensaje en BD			

5.7 **Final de sprint**

Al culminar las dos semanas, el equipo se reunió de nuevo con el cliente y le expuso los trabajos terminados y las dificultades que habían solucionado.

Se volvió a priorizar el backlog introduciendo nuevas historias de usuario y cambios en las existentes.

Y se procedió a hacer otra iteración del sprint con las nuevas tareas o tareas incompletas.

Es muy importante, antes de esto, tener una reunión, sólo del equipo de desarrollo donde se analice el sprint con los problemas que se hayan tenido hasta ese momento, como se han solucionado, que mejoras se pueden introducir para que los desarrollos en el futuro sean más ágiles, etc.

El jefe de proyecto además, actualizará su seguimiento del proyecto, en este ejemplo, se hace con *Microsoft Proyect*, y se generó un informe con el estado del proyecto que se envió al cliente y a dirección.

EDT	Nombre de tarea	Duración	Comienzo	Fin
VHC	Comunicaciones	30 días?	lun 17/06/19	vie 26/07/19
1.1	Desarrollo mensajería	10 días	lun 17/06/19	vie 28/06/19
1.2	Servicio de mensajería	10 días?	lun 01/07/19	vie 12/07/19
1.3	Persistencia	10 días	lun 15/07/19	vie 26/07/19
VHBD	Desarrollo Base de datos	20 días	lun 17/06/19	vie 12/07/19
2.1	Diagrama de base de datos	10 días	lun 17/06/19	vie 28/06/19
2.2	Creación Base de datos	10 días	lun 01/07/19	vie 12/07/19
VHI	Interfaz de usuario	70 días	lun 17/06/19	vie 20/09/19
3.1	Pantalla de visualización	30 días	lun 17/06/19	vie 26/07/19
3.2	Integración en aplicación cliente	10 días	lun 29/07/19	vie 09/08/19
3.3	Integración en la web	30 días	lun 12/08/19	vie 20/09/19
VHM	Desarrollo lógica modelo horno	80 días	lun 12/08/19	vie 29/11/19
4.1	Estudio inicial de los datos recogidos	30 días	lun 12/08/19	vie 20/09/19
4.2	Manejo de resultados	20 días	lun 21/10/19	vie 15/11/19
4.3	Implementación	10 días	lun 18/11/19	vie 29/11/19
VHF	Integración en producción	20 días	lun 02/12/19	vie 27/12/19

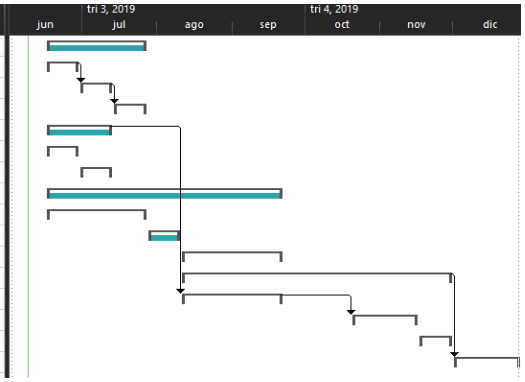


Ilustración 5.11. Estado Microsoft Project (autor)

5.8 **Gestión de cambios**

Los cambios se realizan de una forma ágil. Cualquier desarrollo que va a cliente, se pone en producción y en ese momento se percatan de que no es lo que necesitaban, que vendría bien modificarlo y que además… etc. Simplemente, se redefine o se define una nueva historia de usuario, que se prioriza con el resto de historias y se desarrolla en la iteración que corresponda.

Cuando el cambio implica una modificación de una funcionalidad planificada en el alcance o se introduce una nueva, deberá replanificar el proyecto (en Microsoft Proyect") y hacer un documento donde se describa tal funcionalidad, el responsable de la petición y la fecha, y este documento será enviado a la dirección de Ipsum y a la dirección de Tornillería Emilio.

El jefe de proyecto también estuvo atento a, con el conocimiento del negocio del cliente que adquirió, localizar nuevas oportunidades de negocio, nuevas necesidades en el cliente, que pudiesen dar lugar a nuevos proyectos.

5.9 **Cierre del proyecto**

Es vital saber llegar a esta fase y ejecutarla. Al final de cada iteración se vigila el alcance, y se comprueba en qué estado está el proyecto, para que cuando se detecta que ya se ha cumplido con el mismo y con la calidad acordada, se deberá proponer el cierre del proyecto.

Para ello se envió un informe a dirección y se propuso al cliente una reunión de cierre donde se trató:

- El estado del proyecto, repasando los requisitos iniciales, exponiendo el resultado final ya en producción.
- Los problemas resueltos a lo largo del proyecto.
- Al final de la reunión se hizo un propuesta de mejora o un nuevo proyecto, basado en la necesidad detectada durante el desarrollo del mismo. En este caso se propuso

la inclusión de un modelo que automatizase la gestión de temperaturas del horno, similar al que la empresa ya tenía en otro de sus hornos.

Este acta de reunión, tendría está forma.

Cliente	Tornillería Emilio, S.A.
Proyecto	Visualización horno
Fecha	01/01/2020
Lugar	Tornillería Emilio

Asistente	**Empresa**	**Cargo**
Francisco Hermida	*Tornillería Emilio*	*Responsable IT*
Manuel Suárez	*Ipsum*	*Jefe de proyecto*
Verónica González	*Ipsum*	*Directora departamento*
Pablo Álvarez	*Tornillería Emilio*	*Jefe departamento IT*

Objeto

Estado y cierre del proyecto

Resumen

<Dado el proyecto encargado para la visualización del horno cuyo alcance y requisitos databan de:
....
Se comprueba que están todas las funcionalidades desarrolladas y en producción no detectándose anomalía en las mismas.
Las funcionalidades:
....
No se han desarrollado porque se han modificado debido a lo expuesto en el documento de cambio
....
Además se han desarrollado las funcionalidades nuevas
...
Que mejoran la propuesta inicial y cubren las necesidades del cliente para
....
Ipsum hace costar también que se detecta una mejora importante en este proyecto la cual podría abordarse en el futuro próximo y que constaría de
....>

Además se hace llegar al usuario final una encuesta para recoger el feedback, intentar detectar fallos en el diseño y comprobar su valoración del producto. Esta encuesta se debe hacer llegar a los tres turnos, e intentar que se rellene por parte no sólo del maestro, sino de todos los operarios que usan la aplicación.

Esta encuesta es importante para detectar posibles errores en la funcionalidad, partes no contempladas y que son importantes en su día a día o simplemente para comprobar que el software cumple con las necesidades del cliente.

6 Conclusiones

En resumen, en plena época de la revolución industrial 4.0, la industria necesita seguir modernizando sus instalaciones y automatizando sus procesos para poder seguir siendo competitiva, mejorar su producción, su fiabilidad y su calidad.

Para ello sus departamentos de IT (Tecnología de la Información), necesitan más recursos humanos para poder abastecer de esa tecnología a las instalaciones, esto supone un alto coste para la empresa que muchas veces no están dispuestas a asumir, de ahí que se produzca la subcontratación de estos servicios y crezcan con fuerza las empresas de outsourcing en el sector industrial.

Desde el punto de vista de las compañías industriales, ¿cuáles son las principales ventajas de este outsourcing?:

- Se abarata el coste de desarrollo
- Se aporta valor rápidamente a la fábrica al elaborar en menor tiempo los desarrollos software y su traslado a producción.
- Les permite centrarse en su negocio sin especializar a sus empleados en otro tipo de productos como el software.

Las principales desventajas:

- Posible pérdida de Know-how por parte de la compañía
- Dependencia con la empresa de outsourcing
- En general, software de menor calidad.

Pero sobre todo, hay que destacar el último punto "***menor calidad***" en el pasado hubo intentos por abaratar estos desarrollos a través de empresas implantadas en otros países donde la mano de obra era muy barata, el mejor ejemplo es la India, pero eso demostró no ser un buen modelo por lo que las empresas buscan compañías de outsourcing cercanas que sean más fáciles de controlar y que entiendan mejor sus necesidades.

Desde el punto de vista de las empresas de outsourcing, para poder desarrollar esos productos a bajo coste, necesitan mucha mano de obra, pero al ser de bajo coste se enfrenta a problemas como:

- Poco conocimiento del negocio del cliente
- Mano de obra poco experimentada
- Continua fuga de know-how en la empresa
- Alto riesgo de perder el cliente al estar constantemente negociando los contratos y los proyectos.

En este TFM se estudiaron las principales metodologías existentes, intentando adaptarlas para dar solución a estos problemas de outsourcing de manera que se llegase a unos procedimientos, dentro del ciclo de vida del software, desde que se contrata un proyecto hasta que se integra en producción dicho proyecto, que consiguiesen que:

- Se aporte valor a la industria en un tiempo prudencial, pero sobre todo con un alto grado de fiabilidad y calidad.
- Tratar procedimientos que logren profundizar en el negocio del cliente para dar respuesta a sus necesidades.
- Motivación a los desarrolladores que mitigue la fuga de talento de la empresa y consiga equipos involucrados con su trabajo.
- Crear dependencia con el cliente al dar servicios satisfactorios y marcando diferencia con la competencia en la calidad y no tanto en el coste.

Para ello se comparó como sería la gestión de proyectos desde una empresa de outsourcing dedicada al desarrollo de software industrial, desde un punto de vista de una metodología tradicional Prince2, y desde una metodología ágil como Scrum. Siempre tratado desde el ciclo de vida del software, y su comparación en cada una de las fases.

Tras analizar los pros y los contras de ambas, se concluye que a pesar de tener puntos a favor y en contra, si se trabaja con estas metodologías de forma conjunta, en función del proyecto, aprovechando lo mejor de cada filosofía, se puede obtener un método de trabajo muy sólido que genere productos de alta calidad, fiables y de una forma ágil.

Se comprobó como Prince2 está orientado a la gestión de alto nivel, con procesos más orientados a la parte contractual y de viabilidad del negocio, esto tiene la ventaja de:

- Formaliza los proyectos, propone las herramientas para fijar la comunicación, la apertura de los proyectos, la planificación, el control y el cierre.
- Establece procedimientos claros que consiguen llevar a término el proyecto, siempre controlando que el producto a desarrollar tenga cabida.
- Alto control de los riesgos, con un seguimiento de los mismos.

Desventajas

- No se mete en el detalle del desarrollo.
- Se pierde agilidad en proyectos pequeños donde no es necesaria tanta burocracia.
- No tiene en cuenta a los desarrolladores, no establece procedimientos a la hora de desarrollar los productos.

Por otro lado Scrum se centra en el día a día, en el detalle del desarrollo. Este tiene las ventajas de que:

- Existen procedimientos claros a la hora de desarrollar y analizar el software.
- Involucra tanto al cliente como a los desarrolladores, logrando un compromiso por parte de ambos.
- Da mucho valor a los desarrolladores potenciando su iniciativa y de ese modo su motivación y compromiso con el proyecto.
- Agiliza el desarrollo aportando valor al cliente desde el inicio.

Como desventajas:

- No establece procedimientos para el inicio y cierre del proyecto
- No establece procedimientos para el control y la planificación del proyecto en su conjunto. Gestión de la comunicación, involucrados, riesgos, calidad, etc.

Después de este estudio, es fácil verificar que las dos metodologías son compatibles y como trabajando en conjunto se resuelven en gran medida los problemas anteriormente expuestos.

También se comprueba que no existe una metodología perfecta que garantice el éxito en cualquier tipo de proyecto, sino que este depende en gran medida del tipo de proyecto y del buen hacer del director para ir adaptando la metodología a las necesidades y gestionando los diferentes recursos y stakeholders.

Gran parte de este trabajo está basado en la experiencia de más de doce años en el sector y más de siete dirigiendo proyectos de este tipo por parte de este autor, por lo que tiene un cierto rasgo sesgado de la realidad ya que su experiencia ha sido siempre en proyectos para el mismo cliente.

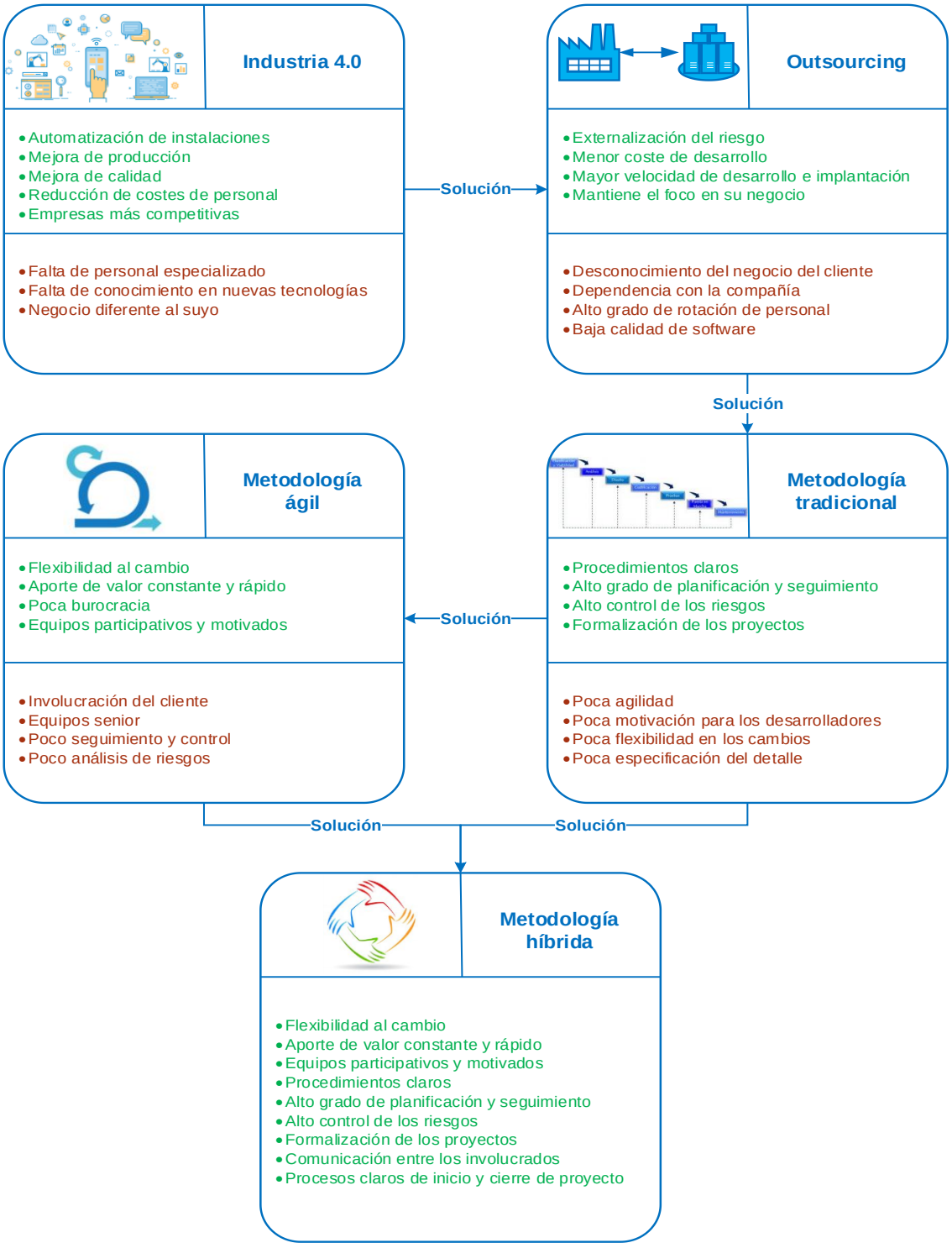


Ilustración 6.1. Resumen conclusiones (autor)

7 Líneas futuras

El software evoluciona constantemente, la tecnología también, vemos como se llaman '*metodologías tradicionales*' a metodologías que no tienen más de 50 años. Además la sociedad avanza, el trato con los diferentes involucrados también es cambiante. Por lo que el estudio para la adaptación de la empresa a este tipo de desarrollos debe de ser una constante.

Como línea de estudio futura, se podría trabajar sobre el uso de metodologías orientadas a sacar el máximo partido a las capacidades creativas de un equipo. Estudiar técnicas de *Designthinking* y ver que utilidad se les puede dar a la hora de desarrollar nuevos proyectos que solucionen problemas en el mundo de la industria de una forma fiable e innovadora.

Adentrarse en el mundo de la psicología de equipos, y ver que técnicas se pueden usar para sacar el máximo rendimiento a los diferentes perfiles. Que técnicas son útiles a la hora de resolver conflictos. Cuáles para llegar acuerdos con proveedores, con los clientes, etc.

Además se podría adentrar en el mundo de la productividad, para mejorar estas metodologías desde el punto de vista de cómo redactar un email, dirigir una reunión, atender una llamada de teléfono o gestionar las interrupciones.

Por otro lado, un tema que puede ser objeto de estudio es la distribución física de los equipos en sus lugares de trabajo. Cuál sería la mejor disposición del equipo en función de la metodología, de cuantos integrantes sería recomendado formar los equipos y que herramientas sería necesario poner a su disposición para agilizarles en lo posible su trabajo.

8 Bibliografía

Adams, S. (2019). *www.gocomics.com*. Obtenido de https://www.gocomics.com/dilbert-en-espanol

agilesurfers.es. (2018). Obtenido de https://agilesurfers.es/tarjetas-crc/

Alexander Menzinsky, G. L. (2016). *Scrum Manager.* Iubaris Info 4 Media SL.

Ambrojo, J. C. (8 de 4 de 2004). India sacude la industria mundial con sus servicios de desarrollo a distancia.

Appelo, J. (2011). *Management 3.0 Practice.* Addison-Wesley Professional.

Baldonedo, J. A. (2017). *Modelo CMMI y métodos ágiles en la gestión de proyectos software.* Oviedo.

Beck, K. (2000). *Extreme Programming.*

Boehm, B. (1989). Software rish management.

Böhm, A. (2009). *Application of PRINCE2 and the impact on Project Management.*

bosco, v. (s.f.). *virtual bosco*. Obtenido de https://virtualbosco.webcindario.com/herramientas.html

Buehring, S. (27 de Marzo de 2019). *Knowledge Train*. Obtenido de https://www.knowledgetrain.co.uk/resources/qualifications/prince2-agile-comparison

Canal, J. M. (2016). *Identificación de procesos para la mejora de los proyectos multidisciplinares y de corta duración.* Universidad de Oviedo.

Conde, J. A. (2018). *disfrutaprogramando*.

Delgado, G. A. (2018). *beeva.com*. Obtenido de https://www.beeva.com/beeva-view/sistemas/cual-es-la-diferencia-entre-unit-testing-tdd-y-bdd/

Delsol, C. (2012). *www.christophedelsol.com*. Obtenido de http://www.christophedelsol.com/10-epic-fails-when-outsourcing-to-india/

Department, S. R. (Enero de 2019). *www.statista.com*. Obtenido de https://www.statista.com/statistics/189788/global-outsourcing-market-size/

Dinngo. (2018). *designthinking*. Obtenido de http://www.designthinking.es

Gestiopolis. (2018). *gestiopolis.com*. Obtenido de gestiopolis.com: http://www.gestiopolis.com/que-es-outsourcing-ventajas-y-desventajas/

Group, I. (2019). *ILX Group PRINCE2 Exams*. Obtenido de www.prince2.com/prince2-exams.asp

Group, S. (2015). *informe Chaos.*

Hoffman, J. (2009). *PRINCE2+agiles Vorgehen=Erfolg.*

How to Build a Great Team. (2006). *Fortune*.

Klein, K. J. (2006). Team Mental Models and Team Performance. *Journal of Organizational Behavior*.

Lledó, P. (2017). *Director de proyectos.*

Martin, R. C. (2012). *Código Limpio.* Anaya Multimedia.

Martínez, P. S. (2013). *Identificación y valoración de técnicas ágiles de gestión de proyectos software.* Universidad de Oviedo.

MDAP Executive Master Project Management. (2019). Obtenido de https://uv-mdap.com/programa-desarrollado/bloque-iii-la-certificacion-prince2/introduccion-a-prince2/

Menzinsky, A. L. (2016). *Scrum Manager.*

Montalván, J. (2009). *Blog de Julio Montalván*. Obtenido de https://juliomontalvan.wordpress.com/2009/03/21/el-outsourcing-en-tecnologias-de-la-informacion-ti/

Navarro, J. G. (2018). *Estudio comparativo de metodologías, herramientas y wiki de soporte para la gestión de proyectos de desarrollo software.* Universidad publica de Cataluña.

Nonaka, T. (1986). *The New New Product Development Game.*

OGC. (2002). *Managing successful projects with PRINCE2.*

Oufaska, Y. (2018). *cafe-agil.com*. Obtenido de https://cafe-agil.com/daily-scrum-meeting-parte-1-2/

Oufaska, Y. (2018). *www.cafe-agil.com*. Obtenido de https://cafe-agil.com/daily-scrum-meeting-parte-1-2/

Pérez, A. (2015). *ceolevel.com*. Obtenido de http://www.ceolevel.com/que-hay-detras-de-la-expresion-gestion-de-los-stakeholders

Pérez, Á. N. (s.f.). *wolfproject.es*. Obtenido de https://wolfproject.es/control-proyectos-agiles-burndown-chart/

Research, 4. (2017). New York.

Ringelmann, M. (1.861).

Schmidt, M. T. (1973). *How to Choose a Leadership Pattern.*

Selectiva. (2018). *selectiva.es*. Obtenido de https://selectiva.es/que-es-outsourcing-beneficios-empresas/

Sevilla, J. (s.f.). *www.overti.es*. Obtenido de http://www.overti.es/tecnologia/299-plantillas-requisitos-xlsx-gestion-y-documentacion-de-requisitos-en-ms-excel

Steel, P. (2010). The Procrastination Equation: How to Stop Putting Things Off and Start Getting Stuff Done.

Suárez, L. C. (2010). *Estudio de la metodología de Gestión de Proyectos PRINCE2: Aplicación a un caso práctico.* Malaga: Universidad de Malaga.

Vyas, T. (2017). *devops.com*. Obtenido de devops.com.

Wittenberg, E. (2006). Programa de Liderazgo de Wharton.

Yetton, V. H. (1988). *The new leadership: Managing participation in organizations.*

9 Anexos

9.1 Anexo 1. Acta reunión en cliente

Cliente	
Proyecto	
Fecha	
Lugar	

Asistente	Empresa	Cargo

Objeto

Resumen

Reparto de responsabilidades y tareas

Tarea	Responsable	Fecha requerido

Fecha próxima reunión

9.2 **Anexo 2. Plantilla para registrar los contactos**

Cliente	
Proyecto	
Fecha	

Nombre	**Empresa**	**Cargo**	**Comentario**	**Teléfono**	**Email**

9.3 **Anexo 3. Control del Kick-off**

Cliente	
Proyecto	
Fecha	
Director proyecto	

	Si	No	Comentario
Designación equipo			
Identificación sponsor			
Objetivos			
Alcance			
Presupuesto			
Plazo			
Riesgos			
Plan de comunicaciones			
Tecnologías a usar			

Comentario:

Cronograma:

9.4 **Anexo 4. Plantilla para toma de requisitos**

Cliente	
Proyecto	

Fecha	
Director de proyecto	

ID	Titulo	Tipo	Descripción	Motivo	Complejidad	Impacto
	<Nombre corto>	*<Funcionalidad>*	*<especificación del requerimiento>*	*<Razón por la que es necesario>*	*<Alto, Medio…>*	*<Alto, Medio…>*

9.5 **Anexo 5. Historia de usuario**

Cliente	
Proyecto	
Fecha	
Responsables	
Objeto	

Descripción

Validación

Prioridad		**Tiempo estimado**	

9.6 **Anexo 6. Plantilla para gráfico de Burndown**

Historia	**Tarea**	**Asignada**	**Esfuerzo** (Estimado)	**Esfuerzo** (Restante)	**L**	**M**	**X**	**J**	**V**	**L**	**M**	**X**	**J**	**V**
<Nombre historia>	*<identificador tarea>*	*<desarrollador>*	*<horas>*	*<horas>*										

9.7 **Anexo 7. Plantilla para validación de interfaz de usuario por parte del equipo de desarrollo**

Cliente	
Proyecto	
Fecha	
Director proyecto	

	Si	No	Comentario
<Funcionalidad 1>	+	−	*<Explicación de porque no cumple>*