Requisitos en contexto

Tabla de contenido

Requisitos

Cada situación requiere su propio enfoque

Hasta ahora hemos hablado mucho de técnicas, de procesos, y hemos visto cómo se aplican estas herramientas en el contexto de nuestro trabajo práctico grupal; y hemos hablado, aunque superficialmente, de qué es lo que pasa con los requisitos, sobre todo, en las metodologías ágiles. Las hemos comparado por ahí también con lo que pasa en entornos un poco más tradicionales o clásicos, si se quiere. Evidentemente, la forma en la cual encaramos las actividades de ingeniería de requisitos va a depender de cada una de las situaciones en las que nos encontremos, cada uno de los escenarios en los que nos encontremos:

No es lo mismo una startup de tecnología que a lo mejor tiene como propósito desarrollar aplicaciones para *smart glasses*[1] y tiene el producto físico pero tiene que encontrar aplicaciones;

No es lo mismo una empresa de tecnología que va a proveer servicios de *software factory*[2] a un banco. Claramente la forma en la cual encaran la ingeniería de requisitos seguramente va a ser distinto y seguramente va a ser muy distinto a la forma en la cual encara la ingeniería de requisitos el cliente. Claramente, hay un trabajo de ingeniería de requisitos que va a tener que estar repartido entre las 2 áreas/partes;

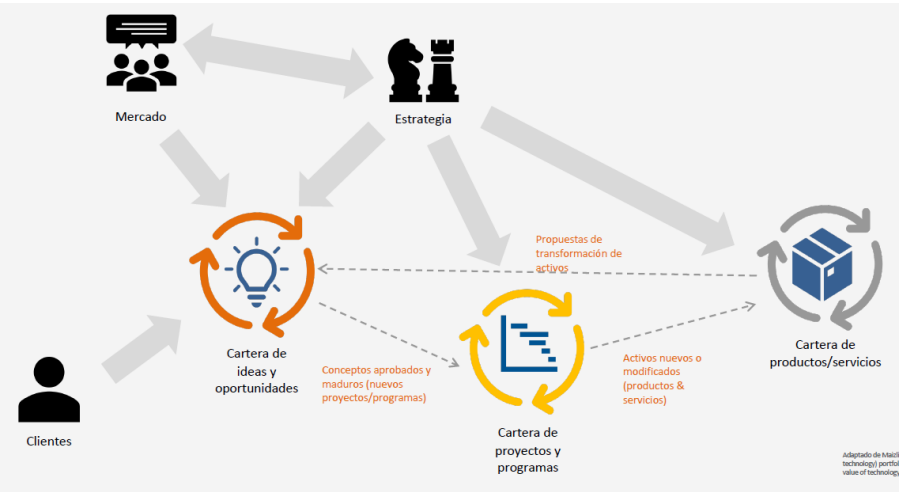
Y otro escenario completamente diferente podría darse si una empresa de tecnología, una empresa de telecomunicaciones con más de 400 aplicaciones desarrolladas internamente quisiera reemplazarlas por un producto o por una aplicación empaquetada. Claramente, la ingeniería de requisitos en estas situaciones sería completamente diferente. Compartirían algunos aspectos, pero en algunos otros términos serían bastante distintos.

Entonces lo que vamos a revisar es cuáles son esos posibles escenarios y qué es lo que podemos hacer en cada uno de esos casos.

Introducción

De la idea/oportunidad al software

Bien, lo que hay que entender de entrada es de dónde sale el software, cómo es que llegamos a desarrollar un software, de dónde partimos, de dónde salimos. Hemos visto muy claramente, sobre todo en el trabajo práctico grupal, que siempre hay alguna elaboración de algunas ideas o de alguna oportunidad de negocios o alguna oportunidad de resolver un problema, eso se elabora y, posteriormente, con un poco de trabajo, llegamos al software, a la solución, al producto.



[1] Los Smart glasses o gafas inteligentes son computadoras portátiles que se colocan en los ojos o en la cabeza y que ofrecen capacidades útiles para el usuario. Muchas gafas inteligentes incluyen pantallas que agregan información al lado o a lo que ve el usuario.

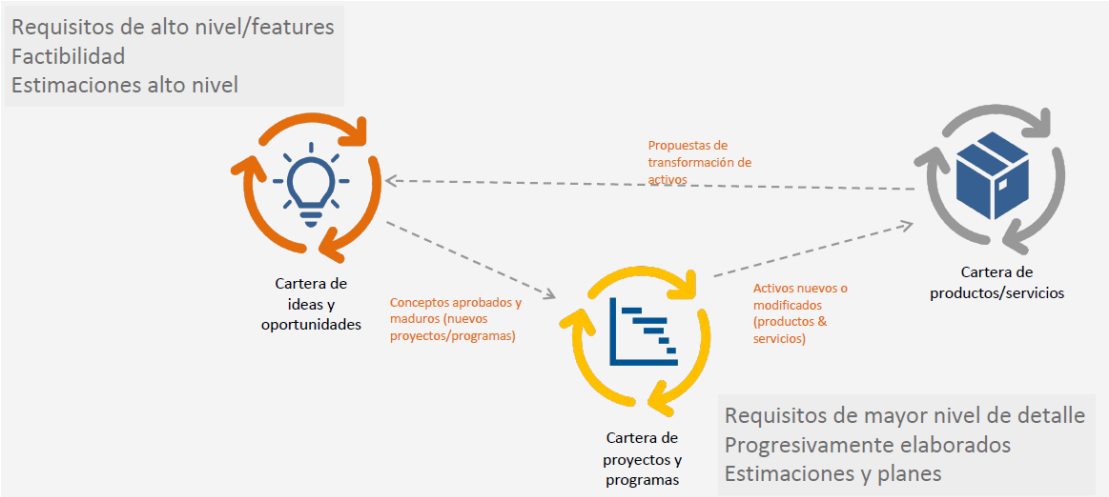
[2] Un servicio de software factory en este contexto es un servicio que consiste básicamente en recibir requisitos, a veces diseños del cliente, y generar software.

Las organizaciones, en general, tienen un plan estratégico que está basado en lo que perciben ellos que el mercado está buscando. La estrategia en definitiva es un plan de juego, es definir cuáles son los objetivos que vamos a perseguir y cómo los vamos a alcanzar. Para poder hacer eso, obviamente tienen que interpretar hacia dónde van las tendencias tecnológicas, hacia dónde va el mercado, qué es lo que quieren sus clientes, qué oportunidades hay en el mercado, qué oportunidades para clientes nuevos. Y esa estrategia en definitiva va a permitirnos identificar ideas para nuevos productos, ideas para nuevos servicios y probablemente oportunidades de mejorar algunas de las cosas que ya tenemos, mejorar no solamente los productos y los servicios, sino también inclusive hasta mejorar la forma en la cual desarrollamos esos productos y servicios. Esas son iniciativas estratégicas que de alguna manera van a alimentar esa cartera de ideas y oportunidades.

[**V** / F] La estrategia es definir los objetivos que se van a perseguir y cómo alcanzarlos.

Adicionalmente a la estrategia, nos va a permitir también determinar cómo gestionamos esa cartera de proyectos, de programas, de iniciativas que vamos a tener ejecución y también va a tener influencia sobre la cartera de productos y servicios que tenemos. Por supuesto que el mercado y los clientes van a ser también generadores de nuevas ideas, nuevas necesidades, nuevas oportunidades. ¿Entonces esto cómo funciona? Si soy una empresa de desarrollo de productos tecnológicos, probablemente yo tenga ese plan estratégico, sepa que este año voy a querer desarrollar una nueva versión de mi producto o desarrollar un nuevo producto, pero también puede ser que yo tenga clientes directos que me estén pidiendo cosas. Entonces, en definitiva, hay que tener esa cartera de ideas, de oportunidades, sobre todo las voy a trabajar hasta el momento en el cual las voy a poder convertir en proyectos. Entonces lo que vamos a ver es que esta cartera de ideas y oportunidades se van a transformar en proyectos, o en programas. Estos proyectos o programas van a generar nuevos activos, en definitiva, nuevas aplicaciones, nuevos productos, nuevas soluciones o, en todo caso, nuevas versiones de esos productos y esas soluciones; y a su vez, el uso de estos productos, de esos servicios, me van a, probablemente, generar propuestas para mejorar o transformar esos productos o servicios que también van a tener que entrar un poco en la cartera.

Si comparamos esto con la situación que están viviendo ustedes en el trabajo práctico grupal, claramente, tuvimos a Juan Zeo que tuvo una idea, la tiró ahí y por supuesto, se empezó a analizar y a elaborar. Eso después se transforma en una iniciativa, en un proyecto, en un programa que va a ser ejecutado y que va a dar por resultado una nueva aplicación, una nueva solución, un nuevo producto. A su vez, si analizamos el caso puntualmente de PSA, nos vamos a encontrar que PSA tiene 2 tipos de proyectos, o 2 tipos de iniciativas: Los que tienen que ver con desarrollar releases, evolucionar releases de sus 3 productos estrellas; y por el otro lado, tienen proyectos que vienen del lado del cliente, que son demanda, de alguna manera, para implementar esos productos en sus respectivas organizaciones. Entonces fíjense que ahí mismo, en esa misma cartera, tenemos ideas para desarrollar nueva funcionalidad que a lo mejor eso viene de marketing y nuevas ideas para nuevos proyectos o demanda de clientes nuevos o demanda de clientes existentes para implementar la solución en sus organizaciones o para hacer algún tipo de customización.



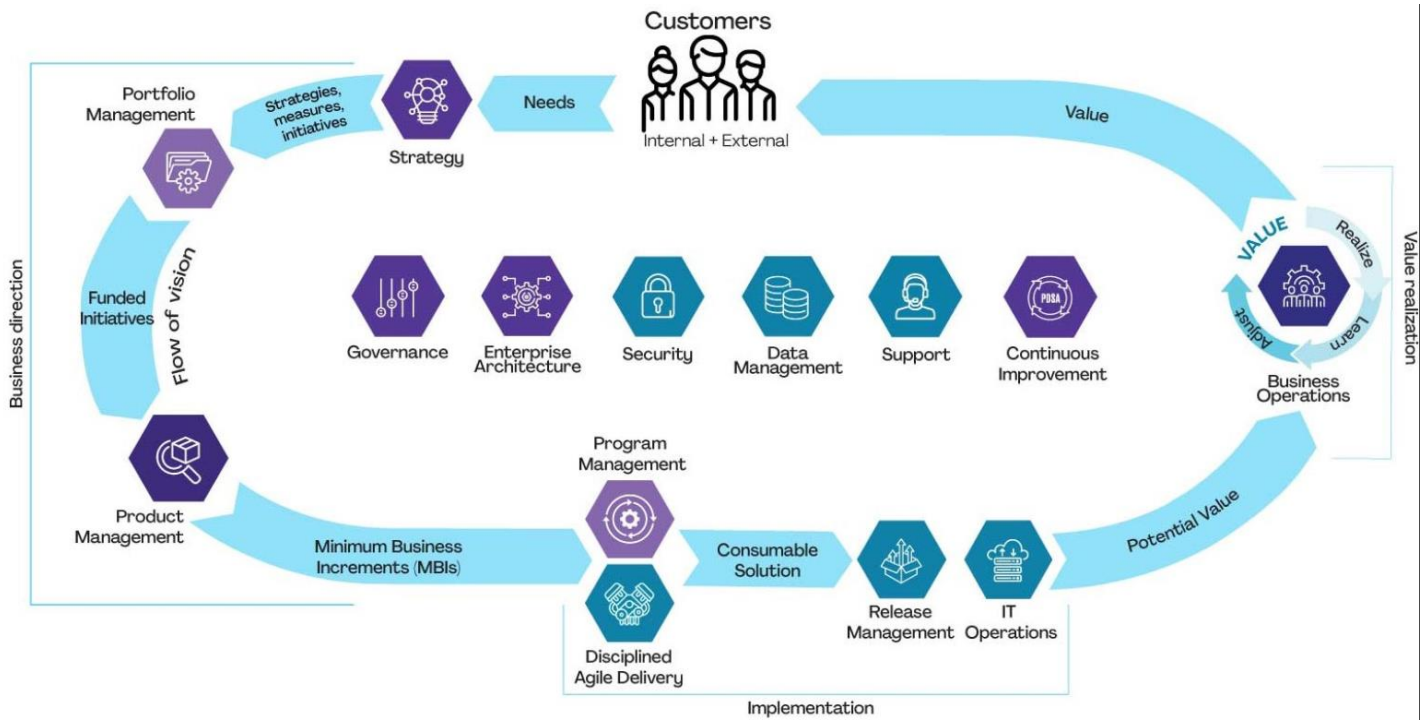
Ahora, todo eso merece cierto análisis, que obviamente no es seguro que esas ideas y esas oportunidades, alcancen la madurez suficiente para transformarse en productos. Entonces claramente esas ideas, que a lo mejor están

esbozadas por ahí a muy alto nivel, van a requerir de una ingeniería de requisitos muy particular. Probablemente no vamos a hacer el análisis de especificación detallada de cada una de esas ideas, sino que simplemente lo que necesitamos saber es, bueno… ¿estamos hablando de 1 millón de dólares o de 1 millón de pesos? Necesitamos poder saber si esas ideas, esas oportunidades, son factibles. Entonces, esto es como si yo les dijera… viene marketing con ideas para el release 2020_2 de nuestro producto con una serie de features, bueno tendremos que analizar si esas features pueden salir o no en nuestro release. Muy preliminar, ¿hay mercado para esas iniciativas? ¿No las hay? Para esas funcionalidades, ¿no las hay? ¿sí, no? Al mismo tiempo, por ahí aparece alguien de preventa o alguien comercial que nos dice… tenemos la oportunidad de desarrollar o de implementar este producto en tal cliente, y ahí nuevamente hacemos un análisis de factibilidad, una estimación, que nosotros llamamos muchas veces de **orden de magnitud**, o sea, necesitamos poder saber si esto que estamos planteando acá es, en terminología por ahí de talle, un XS o XXL. Ahora, si tamizamos eso, las ideas que subsistan, las ideas que tengan una luz verde se transformarán sí en proyectos o en programas o en iniciativas de desarrollo, y ahí sí vamos a elaborar, vamos a trabajar un poco más en detalle en los requisitos. Vamos a necesitar transformar esas features, por ahí esas funcionalidades a alto nivel en requisitos de más bajo nivel: en historias de usuario, en requisitos no funcionales; y eso nos va a permitir trabajar posteriormente en el diseño y en el desarrollo.

[V / **F**] Posiblemente se haga un análisis detallado de todas las ideas y oportunidades para conocer su factibilidad.

Esto puede parecer algo de demasiado nivel, de muy alto nivel, pero tenemos que entenderlo para saber que nuestros proyectos de software no están aislados. Hay otros proyectos que están compitiendo por recursos, y no todas las ideas que tenemos se transforman en productos. Con lo cual, una de las primeras cosas que tenemos que entender es que no todo merece el mismo nivel de análisis. Creo que fue un gurú de la estrategia llamado Peter Drucker, que dijo "…de 1000 ideas para un producto, hay 2 o 3 que se transforman en productos". Entonces, si a esas 1000 ideas les vamos a hacer todo el análisis de requisitos y vamos a estimar cuánto nos va a costar desarrollar esos requisitos, y solamente vamos a terminar eligiendo 2 o 3, que son los más viables, no tiene demasiado sentido. Con lo cual, es una especie de embudo. Es lo que los americanos llaman "funnel". Vamos a ir analizando y estimando etapas en función de la madurez que van teniendo estas ideas.

Ejemplo: DAD



Un ejemplo bien concreto: Ustedes han escuchado de nosotros hablar de DAD[3]. Bueno, DAD plantea un enfoque muy interesante en donde, a partir de las necesidades de los clientes, de las necesidades del mercado, se elabora la

[3] Disciplined Agile Delivery

estrategia; esa estrategia se transforma en definitiva en iniciativas, en mediciones, desde el punto de vista estratégico, que queremos conseguir; eso genera que en la cartera de iniciativas aparezcan estas nuevas ideas para nuevos proyectos, nuevos productos; algunas de esas van a conseguir recursos, es decir, algunas van a pasar el tamiz y van a ser aceptadas; esas que van a ser aceptadas, se van a trabajar, se van a transformar en definiciones de producto, un producto que muchas veces van a ir por una definición de lo mínimo indispensable para que puedan ser utilizados en el famoso MVP[4]; y una vez que tenemos ahí una definición, lo que propone DAD es que se va a ejecutar el programa o el proyecto y así vamos a tener como resultado la solución, el producto, la aplicación, como quieran llamarlo, que se va a liberar, va a comenzar a ser operada, eso va a generar potencialmente valor para los clientes; y, a partir de ese valor generado, los clientes van a poder identificar nuevas necesidades, nuevos requisitos, que en definitiva, van a volver a entrar en el ciclo. Esa es la forma en la cual trabaja una organización. No importa mucho a esta altura del partido, si es una organización tecnológica que desarrolla software o una empresa que desarrolla otro tipo de cosas, no importa. Hoy casi todas las empresas tienen un componente tecnológico importante, con lo cual, si yo defino un producto o servicio de negocio, atrás de eso va a haber seguramente un producto o un servicio tecnológico, con lo cual, el área de sistemas, el área tecnológica, va a estar claramente involucrada.

Startup: Desarrollo de Productos y Desarrollo de Clientes

¿Qué es lo que pasa en una startup? Bueno, en una startup, la verdad que no tenemos muy claro qué es lo que necesita el cliente. Muchas veces tenemos una hipótesis. En el caso que veíamos al comienzo, tenemos un producto de hardware, el smartglass, y tenemos que ver qué es lo que hacemos con eso, qué aplicaciones podemos encontrar. Entonces, tenemos que empezar a establecer hipótesis de qué es lo que potenciales usuarios quieren, y, al mismo tiempo, tenemos que ir viendo si podemos plasmar esas posibles expectativas en un producto mínimo viable. Entonces ahí se dan 2 cosas en paralelo: el desarrollo del producto y el desarrollo de los clientes. ¿Qué quiere decir con esto? Es que voy a tener que ir explorando qué es lo que necesita, qué es lo que entendemos que le podemos vender a nuestros posibles clientes, qué producto es el que se puede elaborar. Elaboramos un MVP y con ese MVP vamos a ver qué es lo que opinan nuestros potenciales clientes y en función de lo que nos digan, refinamos y al mismo tiempo vamos a ir encontrando la vuelta de cómo, por ejemplo, monetizar el producto, algo que a lo mejor no nos planteamos. Ese es el ciclo que siguen casi todas las compañías startup tecnológicas: Tienen una idea inicial para un producto o para un servicio con base tecnológica; desarrollan una versión inicial; lo tiran ahí; ven qué onda con los clientes (o con los posibles clientes); si hay usuarios que empiecen a utilizar la aplicación, empiezan a escucharlos, empiezan a hacer estudios, y en función de lo que digan y el interés, van refinando la definición de ese producto o ese servicio. En las compañías grandes, el escenario es un poco distinto. Hay que pasar por un tamiz y hay que conseguir financiación para desarrollar los productos. En una empresa que está arrancando de 0, la empresa arranca con el producto, con lo cual, claramente, hay que ir haciendo este trabajo en forma simultánea. En una empresa establecida, uno ya conoce a sus clientes, uno ya tiene mayor contacto, sabe qué es lo que puede llegar a necesitar y, en todo caso, si tiene algo, lo van a expresar. Claramente, la ingeniería de requisitos, en este tipo de situaciones, es muy diferente y es mucho más dinámica que, por ejemplo, si estuviéramos trabajando, proveyendo servicios de desarrollo de software para un cliente, como podría ser un banco, como presentábamos en el caso de la software factory del comienzo de la charla.

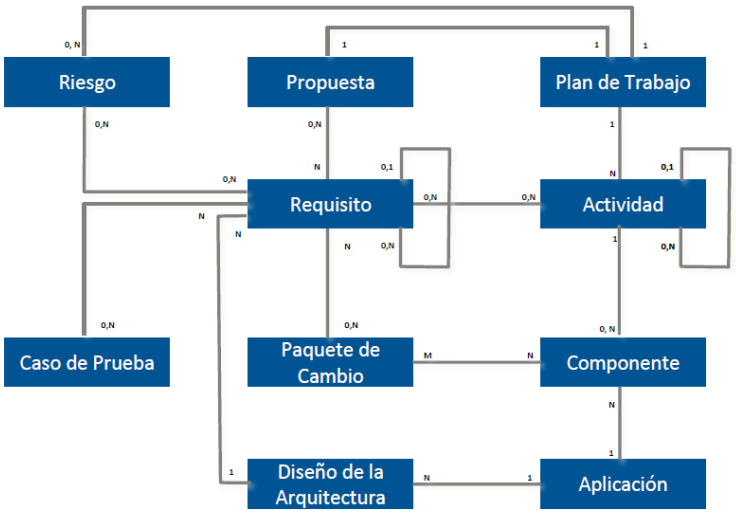
> [V / **F**] Tanto en una *startup* como en una empresa establecida se desarrollan en paralelo tanto los productos como los clientes.

Requisitos en el ciclo de vida

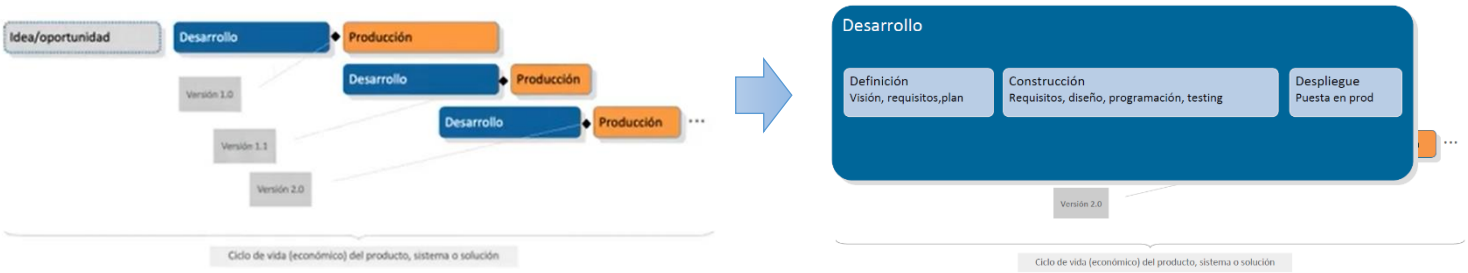
Bien, veamos un poco que es lo que pasa con los requisitos a lo largo del ciclo de vida. Hay una cosa muy interesante que hay que entender es que los requisitos no están aislados. Los requisitos, como ustedes bien saben, son la fuente para un montón de cosas: son la fuente para poder preparar un plan de trabajo, para poder armar una propuesta, poder definir cuál es la visión del producto. Los requisitos tienen asociados riesgos, tienen asociados casos de

[4] Minimal Viable Product

pruebas. Se implementan a través de un paquete de cambio que está asociada a una serie de componentes. Permiten identificar un diseño de arquitectura. Con lo cual, los requisitos son la base para poder hacer todo lo que normalmente uno hace en un proyecto de desarrollo. Hay que tener esto muy claro, cuando nosotros hablamos de trazabilidad, estamos hablando de esto, de poder entender cuál es el vínculo que existe entre los distintos elementos que forman parte de un esfuerzo de desarrollo de software. No solamente es una cuestión de mantener la tranquilidad entre los requisitos entre sí, sino también de saber que, por ejemplo, uno toma una decisión de diseño que tiene que ver con un requisito o uno toma una decisión de diseño que tiene que ver con un requisito no funcional. Eso es sumamente importante.

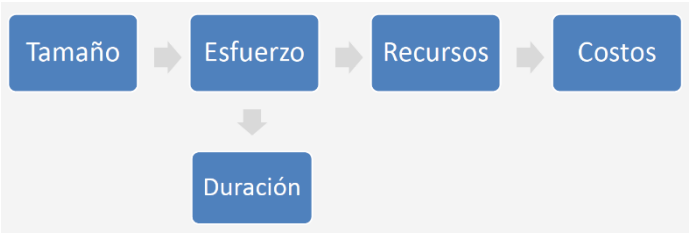


Procesos, métodos, modelos



Lo que también es importante entender es, cómo juegan cada uno de estos elementos que decíamos antes. ¿Qué es lo que pasa cuando estamos trabajando con la idea o la oportunidad (todavía está todo muy verde)? Y, ¿qué es lo que pasa en el desarrollo, cuando empezamos a desarrollar el producto o el servicio? Y claramente ahí también, hay una serie de actividades que tienen que ver con requisitos a lo largo de todo el ciclo de vida.

Estimación



Los requisitos sirven en buena medida para hacer una estimación de esfuerzo. Uno para poder determinar cuánto puede llegar a costar desarrollar un producto se tiene que basar en los requisitos funcionales, en los requisitos no funcionales. En general, todas las técnicas de estimación de esfuerzo, costo y duración empiezan por una estimación del **tamaño**. ¿Qué es el tamaño en este contexto? Bueno, cuán complejo, desde el punto de vista funcional o técnico, es construir lo que tenemos que construir. No es lo mismo desarrollar una aplicacioncita web para registrar las horas que trabajamos, que desarrollar el homebanking de un banco. Claramente son 2 cosas distintas que intuitivamente tienen tamaños diferentes.

Una vez que podemos estimar ese tamaño, estamos en condiciones de poder estimar el **esfuerzo**. El esfuerzo es la cantidad de horas de trabajo, no es la duración. Si yo tengo 2 personas: una persona trabajando durante un mes son 160 horas de trabajo (son 20 días hábiles por 8 horas de trabajo, son 160 horas). Si hay 2 personas trabajando, o si requiero 2 personas para hacer algo, son 320 horas de trabajo. Entonces hay posibilidades, hay técnicas que nos planteamos. Si vos podés estimar el tamaño en una unidad (que ya veremos cuál es) podemos determinar cuál es el esfuerzo. El esfuerzo de la cantidad de horas de trabajo. Y la **duración**, a veces, muchas veces, está en función del esfuerzo. ¿Por qué? Porque la relación entre cantidad de gente involucrada y duración no es lineal. Con esto lo que quiero decir es, si a partir del tamaño yo determino que el esfuerzo para desarrollar lo que quiero desarrollar es de

320 horas, si pongo una persona voy a tardar 2 meses; pero si pongo 2 personas no voy a tardar un mes. La relación entre esfuerzo y duración, o cantidad de personas y duración, no es lineal porque obviamente necesitamos que esas 2 personas se comuniquen, no van a poder trabajar todo en paralelo. Entonces probablemente, en ese caso, la duración va a tener una relación con el esfuerzo que no va a ser lineal. Hay algunas ecuaciones al respecto. Y en función de eso, también podemos estimar qué **recursos** necesitamos y podemos estimar los **costos**. Cuando digo recursos me refiero a recursos humanos y recursos de infraestructura.

[**V** / F] La estimación es necesaria para poder anticipar las posibles magnitudes del proyecto.

Requisitos en el descubrimiento/concepción/inicio

Métodos de estimación

En general, las técnicas de estimación entran en 2 grandes categorías:

- **Algorítmicos**:
  - *Cocomo / Cocomo II:* Es un trabajo que ya tiene alrededor de 30 años. Fue un primer esfuerzo para tratar de determinar, justamente, cuál era el esfuerzo que implicaba desarrollar software. Tener una técnica de estimación más allá de, como decimos nosotros, de los 5 dígitos oscilantes[5]. Tener una estimación basada en algo concreto, en un algoritmo. Parte de una estimación previa que es la cantidad de líneas de código que puede llegar a llevar implementar el producto, o sea, se parte de los requisitos, se hace una estimación de cuántos programas hacen falta, cuántas líneas de código en total, y en función de eso, Cocomo a través de unas ecuaciones y una serie de factores de ajuste permite determinar la cantidad de horas de desarrollo, cantidad de horas de testing, cantidad de horas totales, duración estimada, etc.

  - *Puntos función:* También parte de una estimación del tamaño. La diferencia que la estimación de tamaño no es de cantidad de líneas de código, sino son, justamente, puntos función. Punto de función es una unidad que mide la funcionalidad que tiene un producto de software a construir. Entonces uno, a través de una serie de lineamientos, a través de un algoritmo, puede a partir de una definición abstracta del sistema que podría ser, por ejemplo, las historias de usuario o los casos de uso, identificar determinados tipos de transacciones conceptuales, y a partir de ahí derivar… "bueno esto son 100 puntos función". Y a través de alguna ecuación posterior, traducir esos puntos de función en horas de trabajo.

  - *Use case points:* Se plantea algo parecido a puntos función.

- **No algorítmicos**:
  - *Opinión de expertos:* Que venga alguien que conozca mucho del tema y opine, bueno, a ver esto más o menos puede llevarse tanto tiempo y costarte tanto porque yo lo manejo.

  - *Analogía:* Que es comparar con algún otro sistema similar.

  - *Descomposición:* Que implica armar un plan ya más detallado y determinar cada una de las tareas que hay que llevar adelante y estimarlas.

  - *Story points*

Story Points

Los story points están muy ligados al desarrollo ágil. Los vamos a aplicar al trabajo práctico. Y lo que plantean, básicamente, ¿qué es un Story point? Es una métrica abstracta. Es una métrica que nos muestra cuál es la complejidad relativa de cada uno de los ítems, de cada uno de los requisitos que tenemos en el backlog de nuestro

[5] Se refiere a la seña de más o menos con la mano

producto o en la lista de requisitos. Entonces, no está relacionado con nada en particular, sino que es una medida, si se quiere, subjetiva en donde uno o el grupo de trabajo decide, bueno, a ver, este requisito implementarlo es complejo, muy complejo, extremadamente complejo; o puede decir, simplemente establecemos una escala del 1 al 10, y, a través de algunos mecanismos, el equipo termina de consensuar realmente esa métrica, esa cantidad de story points para cada uno de los requisitos. En el caso puntual, si están usando justamente user stories, uno lo que debería hacer es agarrar las user stories y tratar de determinar, en función de lo que a uno le parece, qué complejidad tiene. La otra es también usar una escala tipo "talle de remera": XS, S, M, L, etc.

Lo que no nos provee el método es una traducción de story points a horas de trabajo. Eso es algo que el equipo va a tener que ir calibrando. Por ejemplo, supongamos que tenemos una serie de historias estimadas, tomamos una parte de esas historias, lo incluimos en una interacción inicial y vamos a tener un story points total que es la suma de los story points de cada una de esas historias. Vemos si eso finalmente se puede desarrollar dentro de la iteración y ahí podemos sacar la relación entre story points y cantidad de horas de trabajo que nos lleva establecer o implementar esos requisitos. Entonces con eso podemos hacer la traducción de story points a cantidad de horas de trabajo calibradas puntualmente a *nuestro* equipo. No es lo mismo una historia con 8 story points en el equipo A, que otra historia con 8 story points en otro equipo. Es una métrica abstracta y muestra una complejidad relativa y es totalmente subjetiva desde el punto de vista de las personas que participan de la estimación, de los miembros del equipo.

> [V / **F**] El valor de referencia entre *story points* y cantidad de horas de trabajo es de **8h** por cada *story point*.

Requisitos en distintos escenarios

Claramente, bueno, ya sabemos en general nosotros, la postura nuestra es que el desarrollo es iterativo es incremental y particularmente hemos estado tratando de aplicar también algunos aspectos, algunas prácticas que tienen que ver más con lo ágil. Pero digamos también, deberíamos analizar qué es lo que pasa cuando uno quiere evolucionar un producto que ya tiene o cuando necesita migrar de un sistema a otro o cuando quiere tercerizar o cuando quiere implementar un paquete o empezar a usar un software como si fuera un servicio. Cada una de esas situaciones requiere un enfoque totalmente distinto o particular.

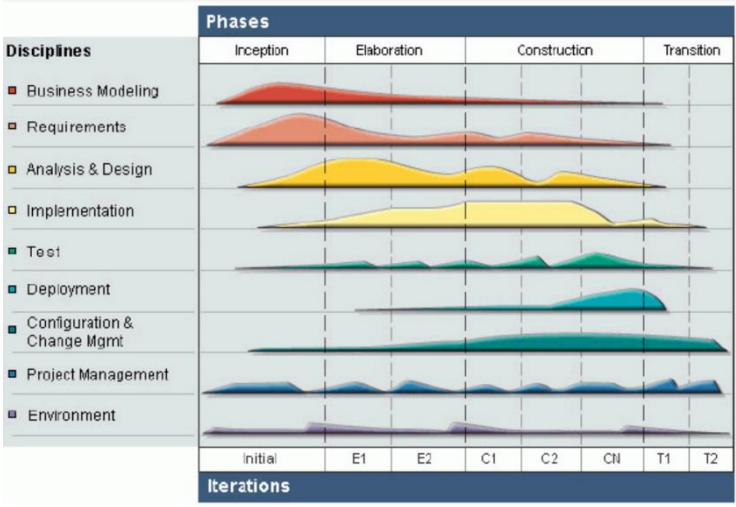
Desarrollo iterativo/incremental/ágil

Vamos a ver qué es lo que pasa en la ingeniería de requisitos en:

- ★ El proceso unificado, que no necesariamente es ágil, pero si es iterativo e incremental (IBM-RUP, Upedu, etc)
- ★ XP: Extreme Programming
- ★ Scrum
- ★ FDD: Feature Driven Development
- ★ DAD: Disciplined Agile Delivery

Proceso unificado

El proceso unificado nace un poco de la mano de UML a fines del siglo pasado, principios de este siglo. Está basado en muchas ideas que andaban dando vueltas por entonces y puntualmente en un proceso llamado "objectory" que era el proceso que impulsaba Ivan Jacobson, uno de los papás de UML. Es claramente un proceso iterativo, incremental. Lo que estamos viendo aquí en forma vertical son justamente las iteraciones. La particularidad que tiene, al final de cada iteración, hay un incremento del producto, muy parecido a lo que pasa con las metodologías ágiles, la diferencia particular

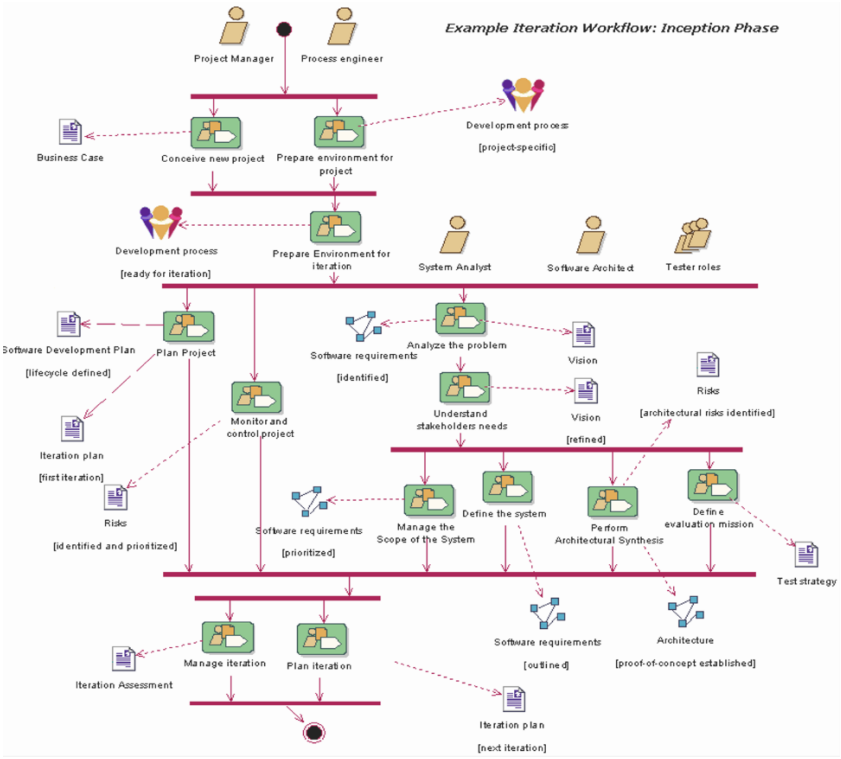


es que RUP decide agrupar a las iteraciones en fases. Tenemos una primer fase de "**Inception**" que es la fase en donde se define la visión y en donde se define el rumbo de la iniciativa y después, una fase de "**Elaboration**" que es más fuerte en todo lo que tiene que ver arquitectura y línea base de requisitos, "**Construction**" y "**Transition**".

Lo que intenta mostrar este gráfico acá es que las distintas disciplinas que forman parte del proceso de desarrollo, ya sea entender el negocio, hacer ingeniería de requisitos, hacer análisis y diseño, programar, testear, etc., suceden con diverso grado de intensidad, dependiendo de la fase en la que estemos parados. Es decir, cada iteración adentro de cada fase pasa por todas las disciplinas pero, claramente, con distinto nivel de intensidad. Así es que estamos viendo acá, por ejemplo, que la parte de requisitos es un poco más intensa en las primeras fases del proyecto, pero no muere del todo, sino que se va atenuando hacia el final; el análisis y el diseño empieza muy tibiamente durante la concepción, pero tiene un pico fuerte en elaboration; implementación que es básicamente programar, empieza desde el principio tenuemente y tiene el pico durante construction; y así con cada una de las disciplinas.

RUP

¿Qué es lo que más o menos suele pasar? La diferencia es que RUP plantea un esquema mucho más elaborado. Primero hay que aclarar algo de Unified Process: Hay distintas variantes. Está la variante comercial, que es la de IBM Rational; hay una variante más educativa que se llama Upedu (Unified Process for Education); después hay Agile Unified Process; hay distintas variantes.



Este es, en particular, el Workflow de una iteración de la versión comercial. Y si se fijan acá lo que vamos a encontrar es una serie de actividades, algunas de las cuales tienen que ver con, justamente, poder entender cuál es el problema que se quiere resolver y establecer la visión del producto. Acá si se fijan tenemos un modelo de requisitos que en el caso de RUP o Unified Process los requisitos se representan a través de casos de uso más requisitos no funcionales. Hay una primera definición de cuál es el alcance del producto que se plasma en un documento de visión, muy parecido a lo que ustedes están haciendo. Un poco la propuesta del Unified Process es bueno, mira, durante la fase de "Inception", que es la primer fase, tenes que tener un pantallazo general de todos los requisitos, por ahí profundizar en un 3/4% de los requisitos que son los que más riesgos te representen, en lo que más pueda llegar a influenciar la estimación de fuerza y duración; y después, posteriormente, ir refinándolo en etapas posteriores. Lo que tenés que poder hacer, y como siempre, es tener una visión inicial del producto, lo suficiente como para poder estimar y establecer un plan, y eso es un poco lo que vemos en esta primer fase[6]. Y por supuesto que esos casos de uso que yo inicialmente identifico durante la Inception, y esos casos de uso que inicialmente identifico durante esta primera fase, se van a ir elaborando posteriormente en el resto de las iteraciones y en el resto de las fases.

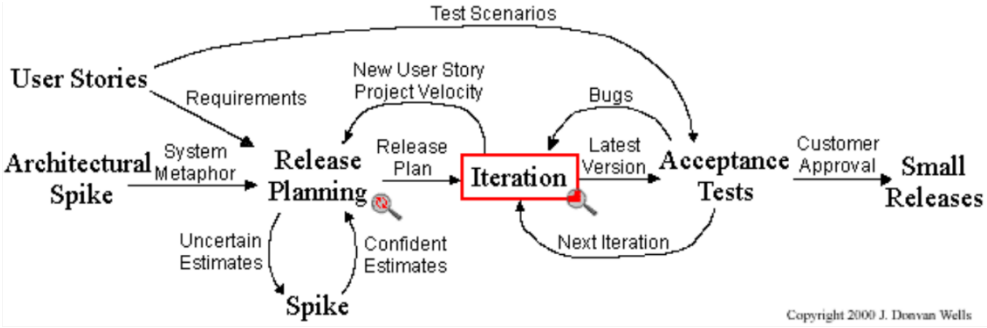
Yo tenía una compañera de trabajo que era muy adversa al riesgo, entonces, antes de darme una estimación de esfuerzo, de duración, de costos, lo que hacía era hacer todo el análisis completo. Entonces, claro, al hacer el análisis completo, no solamente a lo ancho, sino en lo profundo, cuando daba una estimación, estoy en condiciones de ser mucho más preciso. Analicé todo el problema, establecí la solución y me especifiqué todos los casos de uso. Ahora, lo que hay que tomar en cuenta es que del esfuerzo total en cualquier proyecto de desarrollo, yo esta primer fase (la de Inception) no es gratis. Tiene un costo. Normalmente se lleva un 5/10% del total del proyecto. La pregunta es, ¿quién la paga? Porque fíjense si ustedes tienen que presentarle a Juan Zeo su visión del producto que van a

[6] La fase de Inception del gráfico anterior.

desarrollar, lo tienen que presentar en estos días; si Juan Zeo decide que el proyecto no sigue adelante, todo el esfuerzo que les ha llevado hasta ahora es costo hundido. Entonces cuando ustedes estén desarrollando software para un cliente, un cliente externo a la empresa, probablemente esta primera etapa (la de Inception), no siempre la van a poder cobrar. Entonces no tiene mucho sentido que sea demasiado extensa, tiene que ser lo suficientemente larga/detallada como para poder hacer una definición de cuál es el problema, elaborar una solución, pero no irse al detalle de todo lo que hay que hacer porque sino nos llevamos la mitad del proyecto, que es lo que le pasaba a esta compañía de trabajo mía. Si el proyecto les llevaba 6 meses, se pasaba 3 meses para hacer todo el análisis, para hacer una estimación precisa. No tiene ningún tipo de sentido.

> [V / **F**] La intensidad de la disciplina de requisitos en RUP llega a su máximo durante la fase de elaboración.[7]

XP

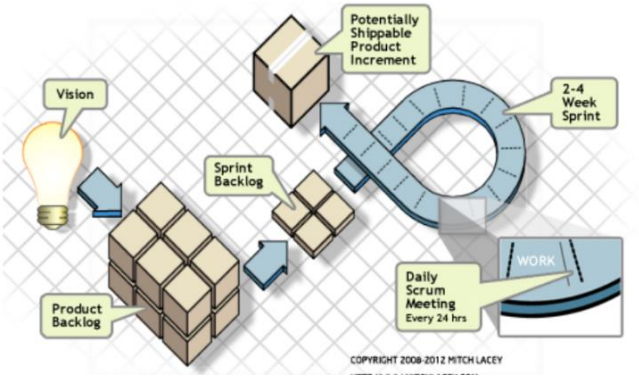


Extreme Programming es una de las primeras metodologías ágiles que surge. La idea de Extreme Programming, como el nombre indica, era… si uno tuviera los distintos aspectos que puedan ajustar en un esfuerzo de desarrollo de software como controles manuales, es como si uno pusiera al palo todos los aspectos: ¿Es bueno hacer programación de a pares? Bueno dale, poné eso al palo; ¿es bueno iterar? Sí dale, iterá muy seguido; ¿son buenas las historias de usuario? Sí, bueno dale, a full con eso. Entonces, lo que plantea XP es, justamente, un enfoque en donde se desarrolla iterativamente, en donde hay determinadas prácticas como programación de a pares, es decir que nos sentamos 2 a programar: mientras uno programa, está usando el teclado, el otro lo está revisando, como una especie de revisión de pares online. Se itera muy seguido, cada 2 semanas; se integra frecuentemente y todo arranca con una definición de cuál es la idea del producto, qué es un poco externo a XP pero que existe antes y que trabaja mucho sobre las historias de usuario y sobre la definición de los casos de prueba. Entonces también confía mucho en un enfoque manejado por test, medio como lo que posteriormente se llamó TDD[8]. Entonces fíjense que a partir de estas historias de usuario, que en definitiva son los requisitos, y la metáfora del sistema, podemos planificar los releases, agarramos esas user stories, decidimos cómo vamos a particionarlas, armamos el release plan y empezamos a iterar. A partir de esas user stories generamos esos casos de prueba que en definitiva son los casos de prueba de aceptación, eso me va a tirar bugs, los vamos a revisar en la próxima iteración. Por supuesto que acá no lo está mostrando, pero se van refinando las historias de usuario en cada iteración. Y en algún momento, cuando el cliente me aprueba, hago el release. En este contexto no hay, si se quiere, actividades formales de ingeniería de requisitos, pareciera como que las user stories están afuera de todo esto. No hay requisitos escritos. Las user stories están en papelitos, están en Post Its, que uno pega por ahí, va poniéndolos acá en la pared. Y si se quiere el eje central acá, está puesto el programar, probar y liberar a menudo. Fue la primer metodología que plantea la utilización de user stories. User stories no es algo que haya surgido con Scrum, que es la otra gran metodología de desarrollo ágil que vamos a ver a continuación.

> [**V** / F] En *XP* es preferible que el equipo sea pequeño y esté muy preparado con un alto nivel de *expertise*.

[7] La intensidad de la disciplina de requisitos (Requirements – gráfico rojo) en RUP llega a su máximo durante la fase de *incepción*.
[8] Test Driven Design

Scrum

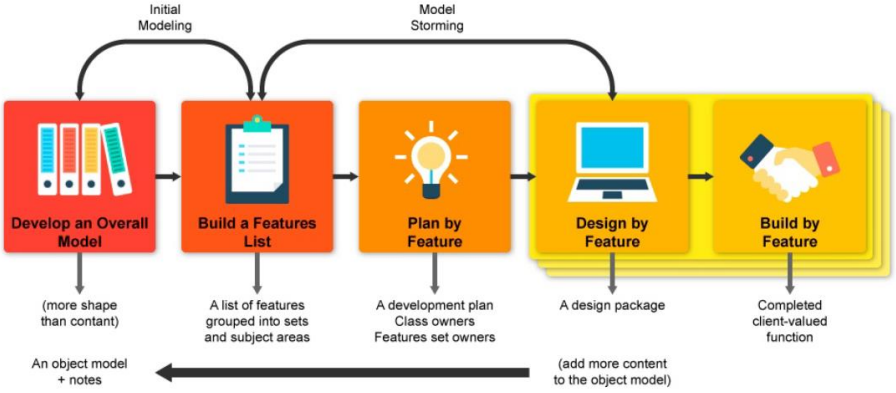


Scrum también es un proceso iterativo, incremental. Plantea una serie de prácticas muy puntuales. Parte de una definición de visión del producto. Eso tampoco está claro en Scrum de donde sale esa visión. A partir de esa visión, uno elabora un product backlog. Ese product backlog en definitiva, describe mediante features cuál es el comportamiento esperado del producto. Algunas funcionalidad van a estar explicitadas más en detalle, mientras las que están más abajo en el product backlog van a estar definidas muy globalmente. ¿Por qué? Porque si están abajo son algo que vamos a desarrollar muy adelante y la vamos a tener que elaborar más adelante. Y lo que plantea Scrum es elaborar un plan de entregas en donde vamos a decir, bueno a ver, este esfuerzo que vamos a encarar lo vamos a desarrollar en pedacitos, en etapas. Cada release implica uno o varios sprints. Un sprint, en definitiva, es un ciclo. Y lo que sucede básicamente es que, el equipo al comienzo de cada sprint toma del product backlog los elementos que van a ser utilizados en el release; decide cuál de esos elementos del product backlog se van a desarrollar en el sprint; hace una estimación; y justamente lo que acá aparece un elemento adicional que es el "Timebox", que también está presente en Unified Process y XP. El "Timebox" es, el equipo es fijo y yo tengo una cantidad de horas fijas. Lo que tengo que ver es, supongamos si el sprint es de 2 personas y dura 1 mes, tengo 320 horas de trabajo ahí. Si los elementos que yo tomo para desarrollar implican un desarrollo que está por encima de esas 320 horas que tengo disponibles, voy a tener que sacarlas. En un proyecto por ahí más tradicional, lo que pasaría es que se agregaría gente al sprint, no es lo que pasa acá. Acá, normalmente hay un Timebox. El Time box es como una caja con duración "en el eje x" y cantidad de personas "en el eje y". Es una caja, es fijo. No puedo agregar más gente. Lo que es variable, en este caso, es qué es lo que entra, qué funcionalidad entra al backlog. Es muy fuerte también este tema de definir los casos de prueba. Es muy fuerte también el tema de automatizar la prueba, y la idea es que al final del sprint yo voy a tener un producto potencialmente entregable.

Si bien Scrum no plantea un esquema basado en user stories, lo que termina pasando es que se utilizan user stories en Scrum. Ustedes ya están familiarizados con esto que es un user story map que nos ayuda a organizar las user stories en releases. Entonces tenemos acá las user stories relacionadas con el Release 1, con el Release 2 y con el Release 3. Scrum no prescribe en absoluto actividades de ingeniería de requisitos, pero lo que normalmente pasa es que, uno durante los sprints, durante las iteraciones, a partir de ese product backlog inicial que tiene todas las historias de usuario, uno los va tomando como recordatorios de conversaciones que hay que tener con el usuario para poder profundizarlas. Y por supuesto que, lo que termina pasando es que, se van a redefinir historias, puede ser que uno particione historias porque no entran en una iteración, en un sprint. Entonces, decidiré algo que inicialmente uno había identificado como una historia la divide en dos o en tres, en función sobre todo de la complejidad que implique. Y lo que sí termina pasando es que el refinamiento del backlog es permanente. Constantemente se va refinando el backlog. Y estos elementos que por ahí están por acá abajo (release 3) medio definidos globalmente, se van, a medida que vamos acercándonos, se van refinando; y lo que va pasando es que decidamos mover historias de release. Algo que teníamos previsto para un release lo adelantemos o lo retrasemos. Entonces acá claramente se ve como los requisitos son el eje fundamental para organizar el esfuerzo de desarrollo. Hay algo que no estamos mostrando acá que es el sprint backlog que en definitiva es identificar cuáles son las actividades que necesitamos nosotros como equipo llevar a cabo en el sprint para poder implementar las historias que hemos asignado a este sprint.

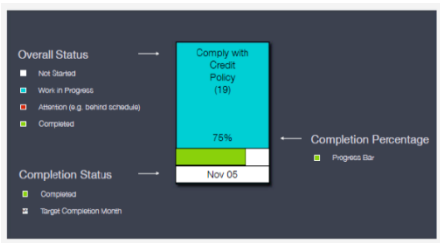
[**V** / F] *Scrum* es independiente de la metodología de desarrollo.

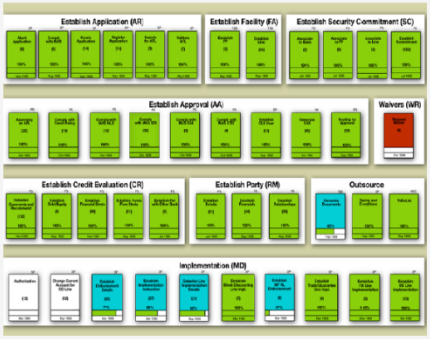
Feature Driven Development (FDD)

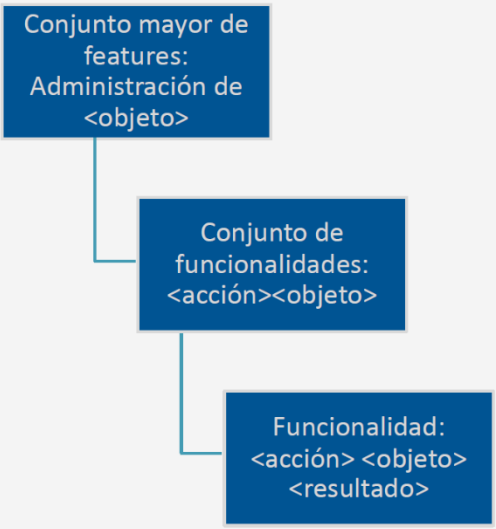


Otra metodología ágil por ahí menos conocida, y para algunos, la menos ágil de las ágiles, es Festure Driven Development o FDD que nace de la mano de Peter Coad, uno de los pioneros en orientación a objetos. Lo que nos propone este modelo es que uno desarrolle primero a partir de una idea del producto, desarrolle un modelo inicial de la arquitectura, desarrolle una lista inicial de features a desarrollar, y que después organice el desarrollo en iteraciones. Dentro de cada iteración lo que se van a tomar es una porción de esas features para desarrollar en una iteración hasta que se agote esa lista de features, por supuesto. Y lo que tiene de particular, y esto sí es notablemente distinto de las otras metodologías ágiles, es que habla de la importancia de diseñar, de hacer modelos y de construir en función de esos diseños, cosa que por ahí ni en XP ni Scrum está explícito. FDD plantea que TENES que diseñar, no es que el diseño es una cosa etérea que pasa en el código, sino que tiene que tener modelos. Esa quizás es la diferencia más importante.

Lo que tiene de lindo FDD es que tiene una estructura para organizar las features que tiene un formato más o menos como este:







La feature, la funcionalidad más baja es quiero hacer tal cosa sobre tal objeto para obtener tal resultado. FDD se basa mucho en el modelado de dominio; uno puede agrupar después funcionalidades en un conjunto de funcionalidades, que se describen como una acción sobre un objeto; y después, en general, un conjunto mayor de features que se especifica como administración de tal objeto.

Ejemplo

Un ejemplo es:

Administración de ventas
- Registrar pedido del cliente → Feature
  - ...
  - Calcular el total de la venta
  - Calcular impuestos
- Emitir factura
  - ...

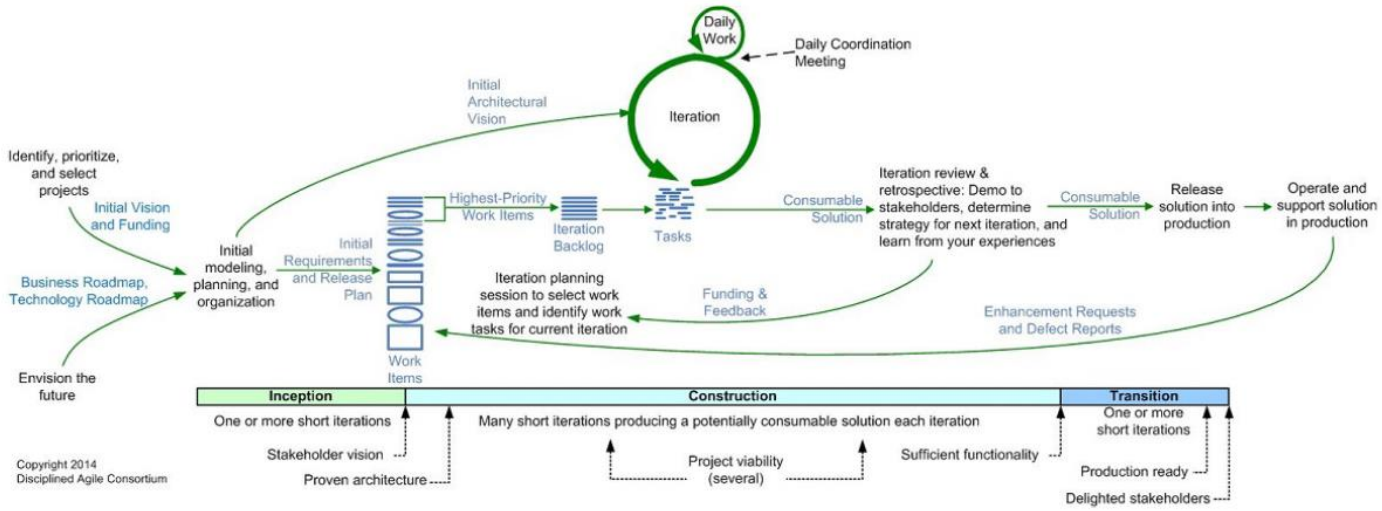
Funcionalidades particulares que tienen que ver con la feature "Registrar pedido del cliente"

Es una estructura de descomposición funcional (un poco parecida al ejercicio 4.10). La diferencia, y lo que es muy interesante, es que se utiliza esta simbología para poder mostrar cuándo va a estar cada una de las features. Fíjense que esta cajita[9] en realidad es algo que es un conjunto de features que nos muestra acá con una barrita qué nivel de desarrollo tiene y cuál es la fecha esperada de entrega. Y acá[10] agrupamos features en conjuntos de features. Entonces acá, en un vistazo general, yo puedo ver cuál es el estado general de nuestro/a producto/iniciativa: lo que está en verde son las features que ya están listas y, lo que todavía está en algún otro colorcito es que son cosas que

[9] La caja del gráfico anterior con fondo gris
[10] Las cajitas verdes que están por debajo del anterior

están trabadas o que están en proceso. Entonces desde ese punto de vista FDD plantea una comunicación visual espectacular de cuál es el estado del proyecto.

Disciplined Agile Delivery (DAD)



DAD es un enfoque híbrido que toma, en realidad, parte de las ideas de proceso unificado (vamos a encontrar acá una fase de "Inception", una de "Construction" y otra de "Transition"; desaparece la fase de "Elaboration") y adentro lo que plantea es algo tipo Scrum o XP. Entonces se desarrolla iterativamente. Hay algunos rituales diarios como plantea también Scrum. Y acá tenemos representado justamente el product backlog[11], que fíjense que justamente las cajitas más grandes, los elementos más grandes, menos granulares, con las cosas que todavía hay que refinar y que se van refinando cada una de las iteraciones que van teniendo lugar durante la construcción; y las que están más chiquititas, más granulares, son las que ya están listas como para poder tomar en el backlog de la iteración y son las que las que permiten identificar cuáles son las tareas que hay que ejecutar en la iteración; y se ejecutan. Entonces la idea es, que al final de la iteración, yo vuelva a revisar el product backlog y refine. Cuando refino, justamente hay cajitas que están con menos nivel de detalle, evolucione y se refine un poco más y halla mayor nivel de detalle. Una vez que yo termino de iterar, justamente llegó al final de la fase de construcción, y ahí ya estoy en condiciones de poner en producción. Y la fase de transición, al igual que el proceso unificado, lo que plantea es una estabilización del código en el ambiente productivo.

En particular, DAD no hace ninguna referencia puntual al estilo de ingeniería de requisitos, aunque sugiere que se usen historias de usuario. Nuevamente en este esquema lo que vemos es una visión general, una identificación inicial del product backlog que está al final del Inception, que constantemente es refinado a lo largo de las distintas iteraciones que tienen lugar acá adentro.

Las fuentes de nuevas ideas u oportunidades de proyecto son:

- ☑ **Demanda del mercado**
- ☑ **Surgimiento o cambios en las normas y regulaciones por parte del estado**
- ☑ **Generación o aparición de nuevos clientes o ventas con los clientes actuales**
- ☑ **Aparición de nuevas tecnologías**

Requisitos en la evolución de sistemas

Los sistema normalmente necesitan evolucionar, difícilmente se queden quietos. Normalmente en las organizaciones hay un flujo constante de demanda para modificar los requisitos, demanda que tiene que ver con nuevas necesidades de negocio, regulaciones, adaptaciones, otros nuevos sistemas con los cuales hay que interactuar y hay que modificar interfaces, muchas veces hay problemas para resolver, hay incidentes que resolver.

[11] Son los "Work items" que aparecen arriba de la interfaz entre Inception y Construction

Desafíos

Entonces normalmente siempre estamos evolucionando esos sistemas; y muchas veces lo que termina pasando es que mucho del conocimiento acerca de esas aplicaciones es implícito, es decir, está en la cabeza de la gente, no necesariamente está exteriorizado/especificado o puesto en algún documento. Esas expectativas que uno tiene de encontrar documentación de sistemas "legacy" (como lo llamamos en la jerga) es eso, es una expectativa, es un deseo. No las van a encontrar. Es muy difícil encontrar algo, y si se encuentra, está muy desactualizado. Entonces hay que hacer un poco de ingeniería de reversa y si uno cae como paracaidista acá y le dicen: "bueno vos acá tenés que mantener tal aplicación" que es lo que normalmente uno hace cuando es muy junior (y cuando es senior también) es tratar de entender qué es lo que hace la aplicación o qué es lo que hace el pedacito de aplicación que uno tiene a su cargo. Va a tener que justamente interactuar y hablar con los que saben. A veces no está muy bueno ir a preguntarle a los usuarios qué es lo que hace el sistema que a uno le acaban de asignar que tienen que mantener, con lo cual hay que tratar de buscarle la vuelta por algún otro lado.

Temas importantes

Si uno cayera como paracaidista… imagínense el caso de las software factory. La software factory que planteamos al comienzo, esa empresa tecnológica acaba de ganar un contrato de servicio de software factory. El servicio de software factory es básicamente, como decíamos antes, es un servicio mediante el cual una organización terceriza la evolución y el mantenimiento de una aplicación. Entonces, uno cae ahí y tiene que hacerse cargo de una aplicación que a lo mejor uno no hizo, entonces lo que uno trataría de buscar es por lo menos entender qué es lo que hace este sistema, al menos, cómo interactúa con el mundo exterior; y para eso, normalmente, lo que uno va a hacer es:

- *Crear algún tipo de árbol de funcionalidades*
- *Identificar* cuáles son los usuarios clave o las *clases de usuarios*
- *Entender los procesos de negocio* que están soportados por esta aplicación
- Tratar de ver cuáles son las *reglas de negocio* y *documentarlas*. Normalmente las reglas de negocio no están escritas, terminan estando implementadas en las aplicaciones. Bueno, no siempre es fácil encontrarle eso;
- Y si uno fuera muy voluntarioso, probablemente trataría de *crear una lista de los principales casos de uso o historias de usuario* que soporta la aplicación. Un poco como hicieron ustedes con el ejercicio 4.11, en dónde había una aplicación que estaba funcionando y tratamos de derivar cuáles eran los requisitos que estaba implementando.
- *Armar el modelo de datos*. Es fundamental para entender.
- A partir de ahí, uno puede empezar a tratar de *plantearse el proyecto de mejora.*

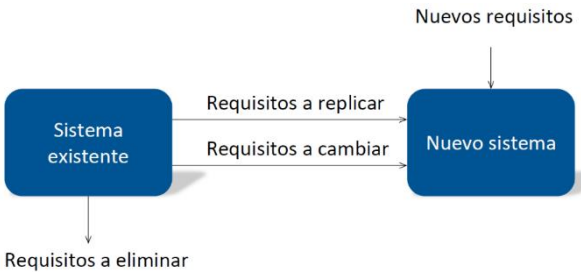
O sea, como que uno hace una especie de Discovery de lo que hay y, a partir de ahí, puede empezar a construir/ trabajar sobre los nuevos requisitos. Ahí hay que tener mucho cuidado del esfuerzo y la utilización que le podamos dar posteriormente a los documentos y a los modelos que podamos armar. Ahí siempre hay un trade-off entre el esfuerzo que implica entender lo que tiene que hacer esta nueva aplicación y lo que potencialmente me puede llegar a beneficiar tener eso documentado y esos módulos creados.

[**V** / F] Suele pasar que el conocimiento de los sistemas es implícito ya que está en los desarrolladores.

Requisitos en la migración de sistemas

Desafíos

Otro tipo particular de proyectos son los que tienen que ver con la migración, que era lo que mencionábamos antes de arrancar. Lo que decíamos es que ese tipo de iniciativas implican que uno toma un sistema que está funcionando; tiene que replicar en el nuevo sistema los mismos requisitos; hay nuevos requisitos que van a estar cayendo y lloviendo permanentemente durante la ejecución de la migración; y adicionalmente, realmente tengamos que revisar, de todas las funcionalidades que tiene el sistema, si realmente vale la pena mantener todo o si vamos a tener que eliminar algunos de esos requisitos que están implementados en el sistema existente o cambiarlos.



Cómo descubrir los requisitos del sistema existente

Todo eso implica un trabajo logístico interesante, un esquema de versionado bastante elaborado. Ahí nuevamente, lo que hay que hacer es:

- Buscar el trade-off[12], buscar el punto intermedio entre no documentar nada del sistema viejo y documentar todo.
- Hay que entender cuáles son los objetivos de negocio para evitar agregar requisitos no prioritarios; yo empezaría, por un lado, tratando de obviamente entender cuáles son los requisitos que está cubriendo el sistema actual para poder reproducirlo en el nuevo sistema.
- Ahí se puede trabajar mucho revisando el código, tratando de armar un mapa de diálogo con las principales interfaces de usuario que tiene la aplicación.
- A partir de ahí, derivar las historias de usuario o los casos de uso principales que estén relacionados con la funcionalidad existente; y al mismo tiempo empezar a documentar las cosas nuevas que hay que implementar, los nuevos requisitos. Era un poco lo que planteábamos antes, los proyectos de migración suelen durar un tiempo y va a haber un momento de coexistencia entre los 2 sistemas y hay que administrar esos nuevos requisitos que llegan. Va a haber que replicarlos en el sistema nuevo y en el sistema viejo también hasta que podamos dar de baja, podamos apagar el sistema viejo. Esa parte por supuesto no es para nada sencilla.
- Nuevamente también es importante derivar las reglas del negocio.
- Es muy importante armar el modelo de dominio.
- Tener un diccionario de datos.

Fundamentalmente, acá lo que hay que hacer es muy prolijo. Nuevamente, hacer un trade-off entre que cosas documentar del sistema viejo, cómo documentar las cosas de lo que vamos a hacer en el sistema nuevo y tener algún tipo de base de datos o de repositorio donde podamos ir poniendo las historias de usuario, los casos de uso principales (el oro viejo) de las cosas nuevas y qué estado es el que tienen.

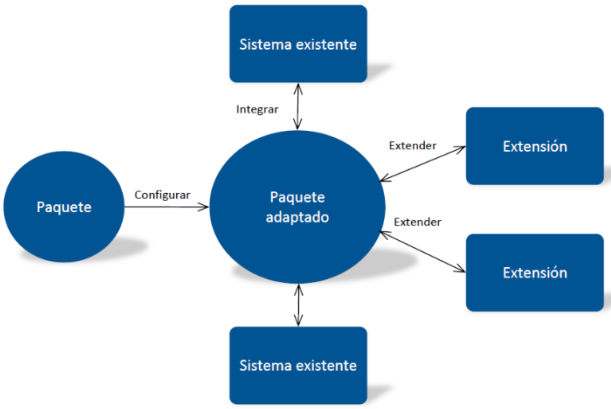
Ante el contexto de un proyecto de migración de un sistema existente, los requisitos:

- ☑ **Se eliminan algunos generalmente de poco valor**
- ☑ **Se incorporan nuevos**
- ☑ **Se cambian en el proceso de migración**
- ☑ **Se replican los más prioritarios**

Requisitos en la implementación de paquetes

Paquetes (COTS: Common/Commercial Off The Shelf) y SaaS (Software as a Service)

Bueno, acá sí nos compramos, tipo PSA, nos compramos un paquete/producto y lo que tenemos que hacer es implementarlo. Es el tercero de los casos/escenarios que veíamos al comienzo. Una empresa con montones de pequeñas aplicaciones que ya no quiere mantener más y que van a ser reemplazadas por un producto/sistema. Un sistema que es naturalmente complejo, que tiene un montón de interfaces, que tiene un montón de parámetros, que van a necesitar interactuar con alguno de los sistemas existentes que seguramente van a quedar, y que va a tener que además, absorber la migración de datos porque vamos a dejar de usar los datos que están en esa serie aplicaciones para empezar a usar los datos que nos plantea este paquete, con lo cual ahí hay seguramente un montón de cosas a regularizar. Y obviamente, hay algunos aspectos que el sistema nuevo no me va a hacer, con lo cual, a su vez, vamos a tener que hacer algún tipo de extensión.



[12] Un *trade-off* es una decisión situacional que implica disminuir o perder una cualidad, cantidad o propiedad de algo a cambio de una ganancia en otros aspectos. En términos simples, un *trade-off* es donde una cosa aumenta, otra debe disminuir.

Alternativas

Este es el escenario clásico de los fines de los 90, y yo les diría que con mucha firmeza durante este siglo el trabajo en Argentina de sistemas era, fundamentalmente, implementar paquetes. Les tengo que decir que año 2001, 2002 y 2003 lo que nosotros decíamos era: ¿qué trabajo vamos a tener para hacer? Porque todos son paquetes. Se implementaban paquetes tipo SAP, Oracle, finantials, había algunos muy viejo JD Edwards, eran todos paquetes que uno traía, configuraba, le hacía alguna cosa custom y lo sacaba a producción. Eran proyectos de años, costosísimos y siguen siendolos, hoy siguen siendo proyectos para nada fáciles, para nada baratos, son proyectos complicados.

Entonces ahí con un producto de software, es el caso de PSA, uno toma la decisión:

- Lo puede usar tal cual viene de acá, lo parametriza y listo.
- Lo puede integrar con otros sistemas.
- O puede cometer el error de pretender de customizarlo, tender a adaptarlo. Y en estos casos, lo mejor que se puede hacer es tratar de ver de adaptarse uno al sistema y no el sistema adaptarlo a la organización, porque suele haber muchos problemas.

Tipos de requisitos

La cantidad de requisitos que uno suele encontrar en este tipo de proyectos es infernal. La lista de requisitos puede llevarse varias planillas de cálculo, varias filas en el repositorio. ¿Por qué? Porque hay temas que tienen que ver con:

- La **configuración** propiamente dicha: Los parámetros. Por ejemplo: qué moneda vas a usar; en qué país vas a operar; qué tipo de producción vas a tener (continua, por lotes).
- La **integración**: Tengo que tener interfaces con algunas otras aplicaciones que voy a dejar funcionando o con la AFIP o con aplicaciones propias.
- Requisitos que yo quiero **extender** para, justamente, cerrar la brecha entre la funcionalidad que me ofrece el producto y lo que yo necesito.
- Temas relacionados con **datos**, y esta es la parte más compleja porque probablemente la estructura de la base de datos sobre los archivos de mis aplicaciones esté totalmente degradado por el uso del tiempo, por el débito técnico, porque no están normalizadas las estructuras, porque puse datos donde no tenía que ponerlos, y traducir eso a un modelo de datos prolijo, lindo y ordenado no es un proceso fácil.
- Y por supuesto que en el medio de todo el proceso de implementación aparece claramente un tema de necesidades de cambio, hay cosas nuevas que hay que hacer (**negocio**).

Evaluación y selección de paquetes

Entonces, la verdad es que son proyectos muy complejos, largos, llevan años, se llevan millones de dólares y se llevan muchas veces el stress de la gente. Acá hay claramente como 2 etapas:

- Evaluación y selección del producto/COTS (Common Off The Shelf);
- Implementación del COTS.

Normalmente la parte de salir a evaluar y seleccionar un producto es un proyecto en sí mismo que tiene su propia lista de requisitos. Normalmente eso implica que uno tenga que desarrollar tablas con requisitos por ahí de alto nivel, que tenga que poder establecerles una prioridad y usar eso como una matriz de valuación contra los N proveedores que uno ve para saber… bueno, a ver… facturación: ¿este producto lo cubre? Sí. ¿este producto lo cubre? Más o menos… Bueno, tenemos que compatibilizar todas esas cosas. Y, por supuesto, no va a haber una solución perfecta a todo esto.

ID	Use Case	Feature	Priority	Tool 1	Tool 1 Comments
1	BA adds new requirements individually	Add new requirement	10	1	
2	BA adds new requirements individually	Automatically create unique ID for each requirement	10	1	
3	BA adds new requirements individually	Document a requirement with rich text formatting	3	0	Only supports text with no formatting.
5	BA models requirements	Describe a requirement with an image directly in the database	6	0.5	Need to do a workaround for this where you link to the resource outside the tool but store the link in the database.
6	BA models requirements	Describe a requirement with an embedded document in the database	8	1	
7	BA links existing documentation to requirements	Link requirements to actual documents in a SharePoint location	4	1	
9	BA adds a bulk of new requirements at once	Batch import structured data as new requirements from Excel	5	1	Batch import is supported and provides support for importing customized Excel files.

Así que esa es una parte, y después, ya durante la ejecución, el problema pasa a ser otro un poquito distinto: ¿Cómo hacer

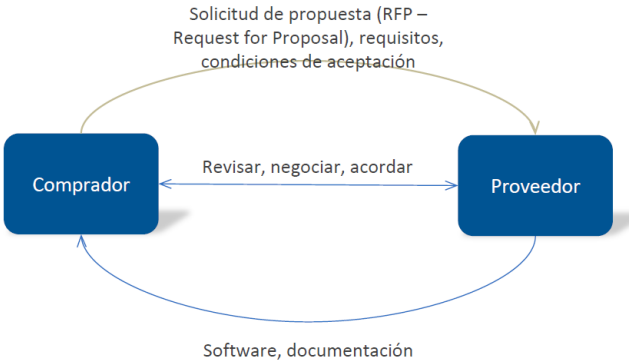
para que, justamente, requisitos de mayor nivel de detalle los podamos en la implementación del paquete? Al igual que en los otros 2 tipos de proyectos, al mismo tiempo que vamos haciendo la implementación del paquete, van a surgir nuevos requisitos. Entonces vamos a tener que canalizar esos cambios también. No hay una solución mágica para esto. Todo implica un esfuerzo de organización importante.

Los tipos de requisitos comunes en un proyecto de implementación de paquetes son:

- ☐ De parametrización
- ☐ De cambio
- ☑ **De integración**
- ☑ **De datos**
- ☑ **De configuración**
- ☑ **De extensión**
- ☑ **De negocio**

Requisitos en la tercerización

La tercerización es la solución mágica de mucha gente de sistemas. "Esto es un lío… Bueno, tercericémoslo. Llamemos a los chicos de análisis de la información que ellos te resuelven todo" Bueno, no es tan fácil. Ojalá fuera así de fácil. Muchas veces decidir tercerizar el desarrollo de una solución de software implica por ahí ir a buscar un proveedor; y probablemente para salir a buscar un proveedor yo tengo que decirle cuáles son los requisitos. No hay manera de que yo pueda transmitir mentalmente cuáles son mis necesidades al proveedor para que el proveedor desarrolle. Tengo que de alguna manera poner por escrito y ahí aparece la necesidad del lado del cliente de especificar los requisitos. Ese famoso ejercicio 4.10, en donde se especifican requisitos usando un esquema un poco más clásico, no hay ni más ni menos que, lo que suele pasar en muchas organizaciones, que salen a buscar un proveedor para desarrollar una aplicación. Entonces, del lado del cliente, está todo el esfuerzo de especificar qué es lo que se quiere conseguir.



Desafíos

Entonces, no es fácil agarrar y decir "bueno che… pará… mirá… hay que desarrollar este sistema. Bueno, se lo tiramos a un proveedor". No es fácil porque hay todo un costo que vos tenés que asumir. En definitiva tenés que hacer una especificación. Si saliste a hacer una licitación, es decir, voy a buscar proveedores y elijo el más barato o el que más me convenga, tuviste que armar un RFP (Request For Proposal). Ese RFP implica que vos tengas que armar los requisitos y eso es un proyecto en sí mismo. Otra es que vos salgas a buscar… y acá hay un tema digamos. Muchas veces yo saco una licitación y el que gana congela el precio, con lo cual, ahí surgen los problemas, porque lo que quieren es: Yo te estimé estos requisitos y te voy a desarrollar esos. Si en el medio cambiamos, o entendimos otra cosa, hay que hacer un contrato nuevo. Entonces, hay que tener cierta cintura para poder adaptarse a esas situaciones de cambio. Actualmente existen otros esquemas un poco más flexibles, en donde uno en realidad lo que hace es identificar grandes features y en el mismo poner algún mecanismo para decir, bueno mirá, el análisis detallado lo haces vos y ahí reestimamos. O por ahí separar el proyecto en etapas: primero un proyecto en donde te compro la especificación y el análisis, y después te compro el desarrollo. Esos son los temas que pueden llegar a aparecer.

Agregamos complejidades, que es lo que pasa cuando el equipo de trabajo no está en mí mismo huso horario ni habla mí mismo lenguaje. Ahí le agregamos una componente más.

Y el otro tema es que el proveedor no siempre tiene conocimiento en el negocio. Obviamente, a veces no lo debería elegir en ese caso, pero muchas veces pasa que a lo mejor el proveedor sí tiene el conocimiento, pero el equipo que asigna el proveedor al proyecto, a lo mejor, no necesariamente. Entonces, empieza ahí toda una bola de nieve que

termina siendo muchas veces desastrosa, porque obviamente, si uno parte de una propuesta llave en mano, con un alcance cerrado, a medida que uno va avanzando con el proyecto ya sabemos que van surgiendo 25 mil cosas. Entonces eso puede estar muy bien para proyectos, a lo mejor, de ingeniería clásica en donde yo tengo todo el diseño completo del producto y salgo a buscar quién me lo construye. En el caso de ingeniería de software es muy complicado porque ya sabemos que hasta el hecho de codificar es diseño. Con lo cual, hay que tener mucho cuidado con la tercerización. No estamos diciendo con esto que no haya que hacerlo. Lo que estamos diciendo, es que hay que ser lo suficientemente inteligentes como para hacer algo que sea interesante.

Temas claves

Entonces, ahí un poco las alternativas son:

- Trabajar en conjunto la definición de los requisitos.
- Plantear etapas.
- Establecer condiciones de aceptación claras.
- Gestionar los cambios a los requisitos.
- Que haya puntos de control en el medio.

Cuando esto pasaba hace 20 años atrás, uno entregaba el RFP con los requisitos, elegía el proveedor y 6 meses después o un año después, aparecía el proveedor con el software, y ya ese software no permitía resolver las necesidades del negocio. Entonces, hay que tratar de plantear un esquema un poquito distinto.

> [**V** / F] En proyectos de tercerización uno de los mayores desafíos al momento de gestionar los requisitos suele ser la falta del conocimiento del negocio por parte del proveedor que ofrece la tercerización.

Conclusiones

Claramente lo que hemos visto es esta charla, es que cada proyecto, cada escenario, cada situación requiere su propio enfoque. No podemos ir con nuestro librito y querer usar en todos lados historias de usuario ni casos de uso, sino que cada situación requiere un enfoque distinto. Claramente el tipo de software a desarrollar, el tipo de relación contractual, los intereses, inclusive hasta el área de aplicación van a tener una influencia muy grande, en el enfoque que adoptemos. No solamente para los requisitos, para el análisis y la especificación, sino también para el proceso de desarrollo en sí mismo.

Cuando estamos en escenarios tipo “green field”, hay más propensión a experimentación. Hay un problema nuevo que hay que entender y que hay que analizar. En cambio, en ambientes/áreas más establecidas/estables, en donde hay que invertir muchos recursos o en donde los riesgos son muy altos, hace falta un enfoque un poco más formal, un poco menos experimental. Es como si yo les dijera: Microsoft con sus productos que a lo mejor están en las primeras etapas, se puede dar el lujo de experimentar y de tener un enfoque de requisitos un poco más laxo. Ahora, cuando un producto ya está establecido en el mercado y… hay que estudiar, no solo hay que tener un proceso de desarrollo mucho más ajustado, sino que también hay que tener mucho cuidado con los requisitos. Hay que tener más cuidado, hay que elaborarlos un poco más, no se cambia alegremente una feature, hay que estudiarlo consensuradamente, hay que analizarlo un poquito más en detalle.

> [V / **F**] En proyectos de implementación de paquetes es común encontrar fácilmente una solución que cumpla con los requisitos más prioritarios.