Modelando

procesos de negocio

Caso de uso: *Ciclo DevOps*

Roberto Frutos Renedo
Agosto 2015

Índice

1. Acerca de mí

Me llamo Roberto Frutos Renedo. Nací un 20 de octubre de 1985 en Valladolid, España y ahora mismo estoy trabajando para ISBAN (Grupo Santander) en el área de Arquitectura Tecnología y Operaciones, en Madrid.

Mi andadura por Madrid comenzó siendo bastante joven, una vez que terminé la carrera de Ingeniería Técnica Informática. Desde ese momento, comencé a trabajar en un sector, el del software, que me apasiona, y que creo que tiene mucho que ofrecer a personas inquietas y entusiastas como yo.

Me entusiasma conocer y dominar cualquier tipo de tecnología, también motivado por la gran variedad de las mismas con las que me he tenido que enfrentar, unas veces por decisión propia, otras por las circunstancias del momento, lo cual me ha permitido acumular una gran experiencia a lo largo de estos años como profesional del mundo del software. Creo que esta experiencia, junto con mi carácter entusiasta por conocer y aprender, son las razones que me han llevado a tratar de plasmar mi experiencia a la hora de diseñar la arquitectura de un gran sistema informático como es en este caso un ciclo de DevOps.

En cuanto a mis aficiones, todo lo relacionado con las dos ruedas. Cuando puedo trato de buscar un hueco para hacer MTB, una de mis pasiones desde que era bien pequeño y mi padre me compró mi primera bicicleta, desde ese momento no he parado. Además de la bicicleta, me encantan las motos, hasta hace un año practicaba moto-cross, pero por diversas razones lo fui abandonando y finalmente vendí la moto. Ahora estoy buscando una nueva, así que pronto retomaré esta excitante afición.

Por otra parte, creo que soy un poco adicto a los retos, y así como hasta el momento había dejado de lado el aprendizaje de otras lenguas, el año pasado decidí apuntarme a la Escuela Oficial de Idiomas, y aunque nunca haya sido una de las materias que me haya apasionado especialmente, conseguí superar el primer curso de intermedio (B 1.1), por lo que si todo va bien este año conseguiré el certificado B1.

Como con esto parece que aún tengo algo de tiempo libre (esto no lo piensa así mi pareja), estoy trabajando/estudiando con el objetivo de conseguir la certificación en ArchiMate 2.1 antes de retomar la Escuela Oficial de Idiomas.

Este es un extracto de cómo soy, mis intereses personales, aficiones y carácter.

2. Motivaciones

Como en primer artículo dejé entrever que escribiría un segundo, intentando plasmar el detalle que había logrado conseguir del cliclo de DevOps mediante BPMN 2.0, ya estaba prácticamente obligado a molestaros de nuevo con otro artículo. Asi que, bien, ahí está la razón principal. Por otro lado, parece ser que tengo más tiempo libre de la cuenta, lo que, unido a lo anterior, da lugar a este segundo documento. Espero te resulte ciertamente interesante y puedas obtener algo de valor del mismo.

En este segundo artículo, que he evitado nombrar como la segunda parte del primero, por aquello de *"las segundas partes nunca fueron buenas"*, trataré de comentar el detalle de los procesos clave que componen DevOps, sus actividades más importantes y las consideraciones de mayor interés a tener presentes, todo ello mediante el uso del lenguaje estándar BPMN 2.0.

Si no has podido leer mi anterior artículo acerca de cómo modelar una gran arquitectura, tomando como base el caso práctico de un circuito de DevOps, aquí te dejo un referencia al mismo, por si diera la remota casualidad de que te sintieras interesado en el mismo.

- Modelando una gran arquitectura empresarial
  https://drive.google.com/open?id=0BymCGWR0IjzkbVY1MFlpaF9paE0

Cualquiera que tenga acceso a este documento puede contactarme con el objetivo de transmitirme sus valoraciones, críticas o comentarios acerca del mismo en la siguiente dirección de correo electrónico: robfrut@yahoo.es

3. Pero...esto ya está modelado, ¿no?

Cualquiera que comience a leer este artículo, y no tenga un extenso conocimiento de los lenguajes de modelado estándar, puede pensar que, habiendo diseñado nuestra arquitectura con ArchiMate 2.1, ¿por qué modelar de nuevo el ciclo DevOps con otro lenguaje diferente? Bien, es una duda razonable y la respuesta es sencilla: modelando los diferentes procesos de negocio que intervienen en el ciclo DevOps con BPMN 2.0, conseguiremos un perspectiva más detallada del flujo de procesamiento, orden de las tareas, sincronización de las mismas, ejecución paralela de subprocesos... Detalle que, como mostraré en otro documento, es clave para entender DevOps y poder llevar a cabo una implementación y configuración adecuada de los diferentes procesos que lo conforman. En definitiva, poder montar un circuito de DevOps con ciertas garantías.

Ante esta situación, nos puede venir a la mente algún tipo de modelado de flujos, de definición de procesos negocio, diagramado de tareas... Tomando como base la experiencia de haber recorrido ese camino, recomiendo abandonar esa idea rápidamente. Es sencillo, hay un lenguaje estándar definido y diseñado para tal objetivo, usémoslo.

Como herramienta para modelar el sistema en BPMN 2.0, en mi caso, he utilizado Eclipse con un plugin adicional que permite modelar con el estándar BPMN 2.0. Existen otro tipo de herramientas, seguramente mejores y peores, no entraré en un debate que podría no tener fin acerca de cuál es mejor o peor, solamente comentar que mi experiencia con el componente de Eclipse ha sido satisfactoria y ha cumplido con los requerimientos a la perfeccción. Tanto si ya estás acostumbrado al uso de Eclipse u otro tipo de IDE, como si decides probar esta herramienta por primera vez para este fin, no te será complicado su uso. Te dejo aquí la URL del componente.

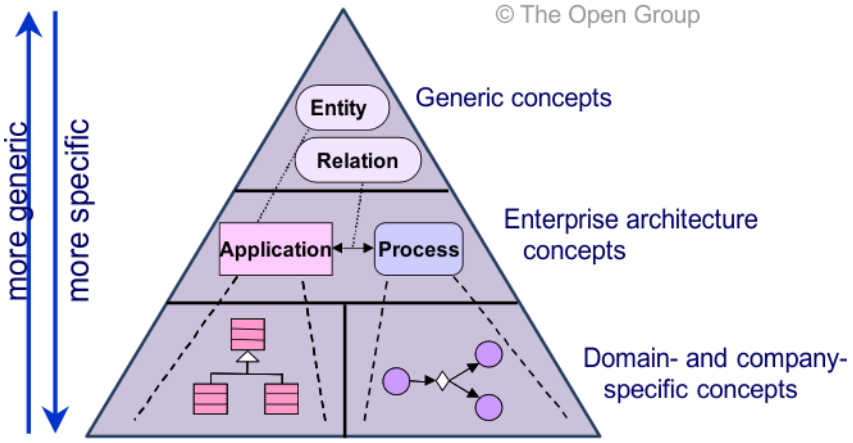
- BPMN2 Modeler
  https://www.eclipse.org/bpmn2-modeler/

Si decidieras probarlo y deseas transmitirme tu experiencia o cualquier comentario acerca del mismo, puedes hacerlo escribiéndome a la dirección de correo que he dejado en el apartado anterior (aún siendo para recriminarme el tiempo perdido).

4. ArchiMate, BPMN, UML

Antes de comenzar a ampliar el detalle de nuestra arquitectura, es importante que tengamos claro qué nos puede ofrecer BPMN y UML con respecto a ArchiMate, asi como la relación entre estos lenguajes dentro de la visión global de una arquitectura compleja.

En este apartado, me gustaría mostrar cuál es la relación que hay entre los modelos de arquitectura que mostré en mi artículo anterior (si no has tenido oportunidad de leerlo, básicamente se muestran algunas de las vistas mas importante de la arquitectura DevOps, modeladas con ArchiMate 2.1) y los procesos de negocio que se presentarán más adelante. Esta es una figura que ilustra a la perfección lo que os quiero transmitir (obtenida de la especificación de ArchiMate 2.1 que podeis encontrar en la red).



Metamodels at Different Levels of Specificity

En esta imagen, podemos ver los diferentes niveles de detalle que nos ofrecen los lenguajes estándar de modelado y la relación entre ellos. Podemos ver claramente, qué nos aportará BPMN en todo esto sin necesidad de extenderme mucho más.

Con respecto a UML, en el caso de uso que estoy mostrándote, no le he encontrado mucho sentido a su aplicación, ya que nuestros componentes son productos o software que no hemos desarrollado nosotros mismos y que por tanto, para nosotros son una “caja negra” en cuanto a diseño de arquitectura de software se refiere.

5. A tener en cuenta

Quiero que, antes de continuar, veamos unos aspectos que debemos tener presentes durante la lectura de este documento y que están más relacionados con DevOps que con BPMN.

Acerca de la organización del proyecto de software, lo más importante que habría que mencionar es que dicho software, en la fase de construcción, debe dar lugar a diferentes artefactos, cada uno relacionado con un aspecto concreto dentro del proceso de DevOps:

- Binarios de software – nuestro software en cuestión
- Pruebas de integración
- Pruebas funcionales
- Pruebas de aceptación
- Pruebas de calidad (rendimiento, seguridad, carga...)

Además de los componentes de pruebas que se deben generar fuera del proyecto de software propiamente dicho, no debemos olvidar que éste debe de contener sus pruebas unitarias y cualquier otro tipo de pruebas que se haya considerado de importancia para la aplicación, es lo que más adelante veremos en alguno de los modelos de DevOps como “pruebas de aplicación”.

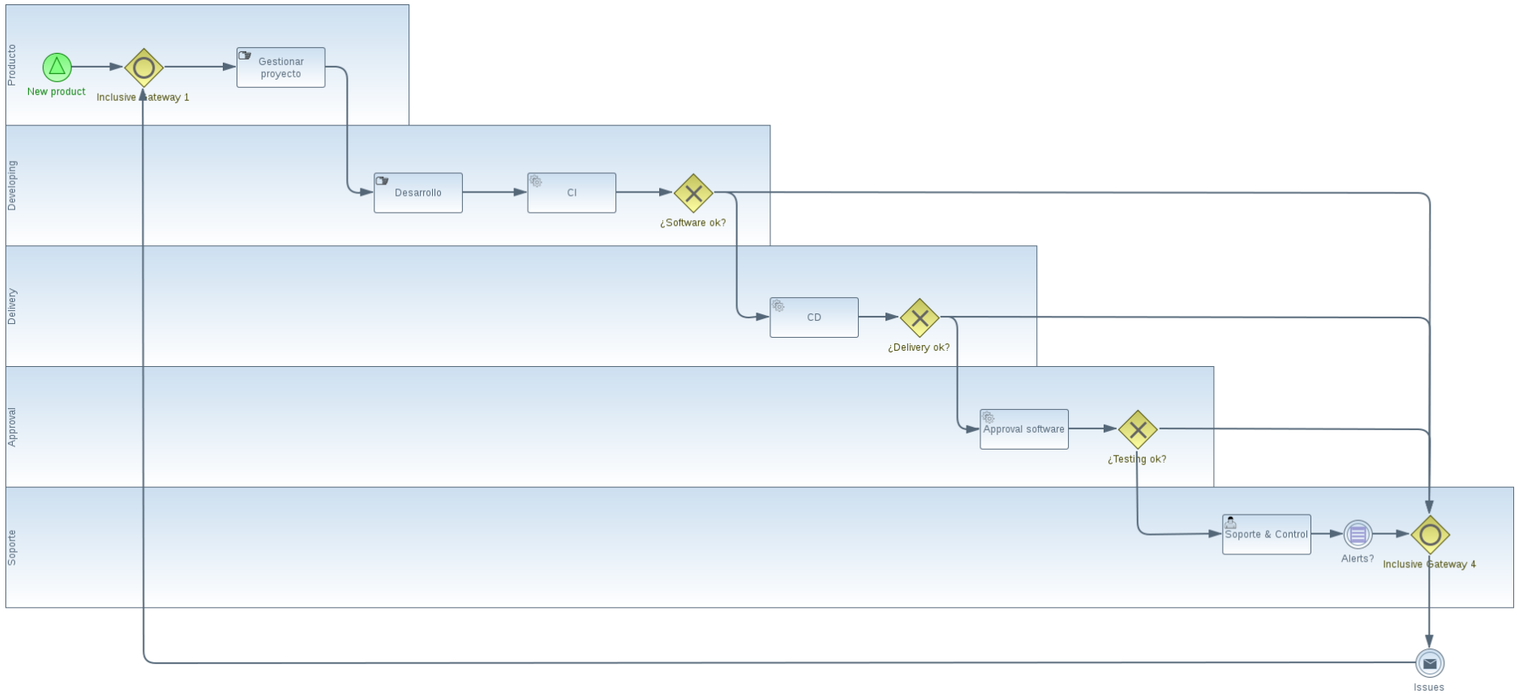
He de aclarar que todas estas pruebas puede que no apliquen a todo tipo de software, pudiendo darse el caso de que exista una aplicación en la que no tenga sentido realizar algún tipo de prueba. Lo importante en todo esto es que existan, del tipo que corresponda en cada caso y siempre aplicando el sentido común. No debemos ser excesivamente rígidos en este aspecto, o nuestro DevOps se verá enfrentado a un cierto rechazo por parte del equipo de desarrollo.

Es importante también mencionar que un proceso como DevOps, implica tener definida la infraestructura de la aplicación de alguna manera junto con nuestro software, en ese sentido, podemos requerir que cada proyecto tenga una carpeta predeterminada donde aloje la definición de la infraestructura que necesitará para poder funcionar correctamente.

Es común que este fichero de definición esté en formato YAML, formato que se está convirtiendo poco a poco en un estándar defacto para la configuración de aplicaciones y software en general. En diversos artículos de la red, podréis encontrar más información al respecto, es un concepto conocido como “Infraestructura como código fuente”. Este concepto, es una de las claves de todo el proceso DevOps.

Puede que no hayas percibido la importancia de esto último, detente un minuto a pensar lo que supone DevOps para el desarrollo de un software particular. Vamos a construir un software, lo vamos a ejecutar, vamos a lanzar las pruebas necesarias dejando un registro de los resultados y por último, si todo está bien, dejaremos nuestra aplicación funcionando en un entorno concreto. Todo esto sin intervención manual, de modo automático.

6. DevOps como proceso de negocio



Este, podríamos decir que es el mapa global de los procesos que componen DevOps, desde mi punto de vista, claro está. Quién más quién menos, ha trabajado con alguna herramienta de gestión de tareas (Gestión de proyectos), sabe lo que es el desarrollo de software (Desarrollo), ha oido hablar de integración continua (Continuous Integration CI), ha soñado con la entrega continua (Continuous Delivery CD) y ha tenido que pelearse con el área de soporte y mantenimiento (Soporte y control).

Este modelo nos puede ser de gran utilidad a la hora de tener una visión global de todo el ciclo DevOps y poder transmitir nuestro pensamiento a otras personas interesadas o involucradas en esta nueva forma de hacer software. Creo que es bastante claro y simple de entender.

Puede que te llame la atención una fase que he denominado aceptación del software (Approval software), creo que es de invención propia, no así aquello que engloba la misma, que no es otra cosa que la ejecución de los diferentes tipos de pruebas asociadas al software que queremos construir. Por razones de desarrollo, de mejora de rendimiento, organización de código fuente y por ser un caso bastante claro donde podríamos aplicar procesamiento paralelo y ahorro de costes, he decidido que estén fuera de los procesos de integración continua y entrega continua.

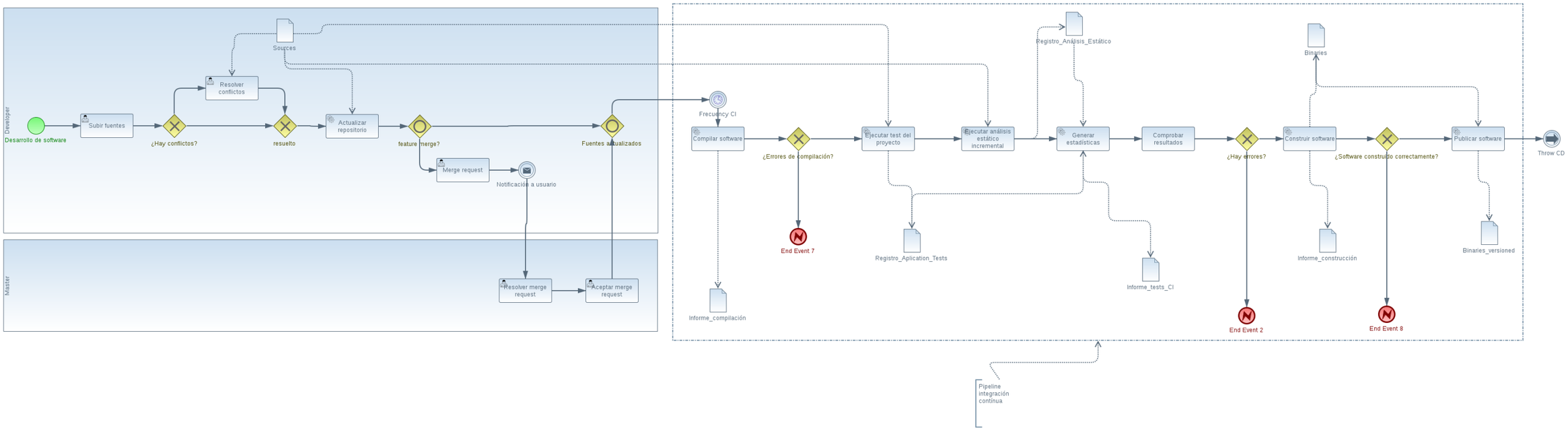
La clave en este punto es entender qué procesos hay en DevOps y cuáles son susceptibles de ser automatizados mediante herramientas o algún tipo de desarrollo personalizado. En este sentido, a grandes rasgos, podemos identificar tres procesos de negocio en el modelo que nos pueden ser de interés:

- Integración continua (versionado, construcción, pruebas, publicación de artefactos)
- Entrega continua (despliegue, notificación de entorno)
- Aceptación del software (pruebas, notificación de aceptación)

Habrás notado que no he mencionado la fase inicial de gestión del proyecto, es una fase en la que se definen los requisitos y especificaciones del software (historias de usuario o tareas), se priorizan las funcionalidades, se definen criterios de aceptación y es el punto desde el que los desarrolladores suelen partir para comenzar el desarrollo de software. Como podrás apreciar (si observas el modelo) es una fase manual y en la que poco podemos hacer para su automatización. Por las mismas razones, tampoco entraremos a detallar la fase de desarrollo de software y tampoco la de soporte y mantenimiento, fases manuales (observa el modelo).

A continuación, mostraré el detalle de los procesos clave, evitando mostraros las fases en las que de cara a la automatización no aportamos demasiado, ya que considero de mayor interés y complejidad las otras.

7. Integración continua



No es mi objetivo relatarte por escrito todo aquello que está en el modelo e intentes comprender por completo lo que ahí se ha modelado, sino que percibas que BPMN puede ser de gran utilidad para reflejar cómo funciona un proceso de negocio complejo y de grandes dimesiones como es DevOps.

En este proceso he intentado dejar constancia de la situación producida cuando, como desarrollador, quiero realizar una subida de código fuente a un repositorio. Como puedes deducir del modelo, el sistema de control de código fuente debe de ser colaborativo, permitir diferentes ramas, disponer de varios roles... puede que ya estés pensando en uno concreto, muy de moda últimamente, pero estamos en una fase de diseño del sistema y por tanto, debemos mantenernos al margen de cualquier contaminación tecnológica. Es la primera parte del proceso, donde intervienen el desarrollador y el responsable de la rama principal.

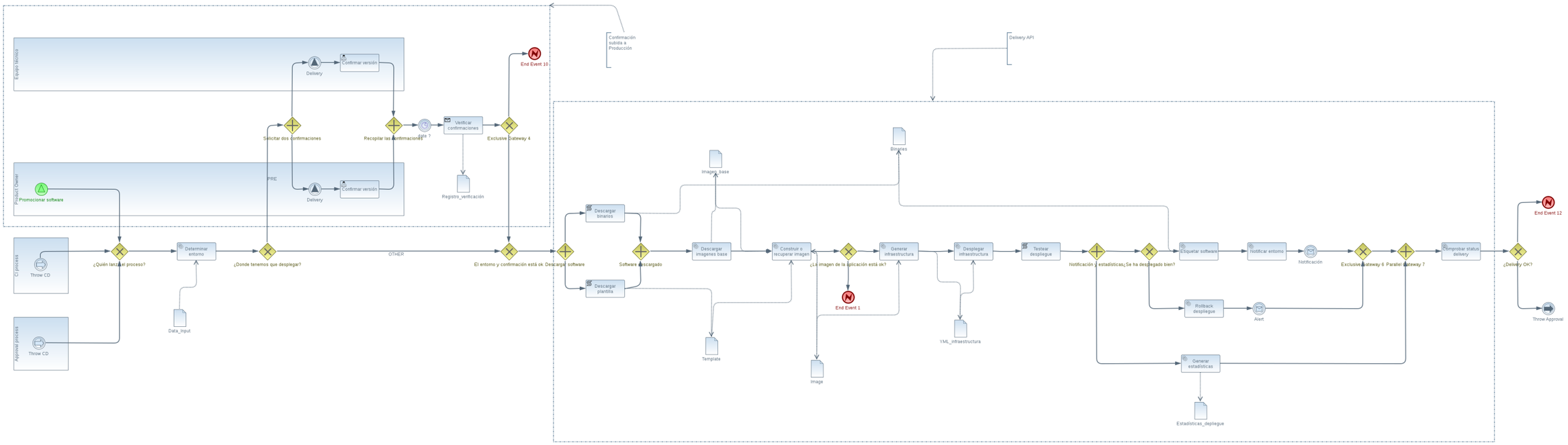
Si eres usuario de cualquier software de gestión de fuentes, notarás una ausencia de diversas acciones que como usuario puedo hacer. No te avalances sobre mí por ello, soy consciente, no he querido entrar a modelar acciones como la descarga de fuentes, crear una nueva rama... ya que creo, son acciones propias del sistema de fuentes y no tienen incidencia sobre el proceso DevOps.

Por otra parte, también podrían detallarse más tareas como “Resolver merge request”, “Resolver conflictos” u otras, es algo que estoy percibiendo en estos momentos. En primera instancia, intentaré hacerlo en otro modelo diferente intentando no dar pasos hacia un modelo de gran complejidad y sí hacia varios más simples y legibles. Podríamos afrontar esto en una fase posterior, ya que son tareas con un alto componente de participación del usuario. En esta fase de la arquitectura lo importante es que queden reflejadas en el modelo actual.

A partir del momento en que ha sucedido una actualización de fuentes, es el momento en el que se disparan las tareas remarcadas dentro de “Pipeline integración continua”. He destacado esas tareas de este modo por considerarlas automatizables mediante algún tipo de herramienta, proceso o sistema de software. Puedes identificar que se ejecutarán tareas como la compilación del software, la generación de estadísticas o la publicación de los artefactos.

Es importante que no pases por alto la presencia del último evento del proceso “Throw CD”, es decir, “lanzar entrega continua”. Es el modo en el que reflejo la conexión entre la fase de integración continua y la fase de entrega continua, entre un modelo BPMN y otro modelo BPMN diferente.

8. Entrega continua



Es evidente que el nombre anglosajón suena mejor y lo reconocerás más fácilmente, estoy hablando de Continuous Delivery, pero intentado en la medida de lo posible ser fiel a un mismo lenguaje.

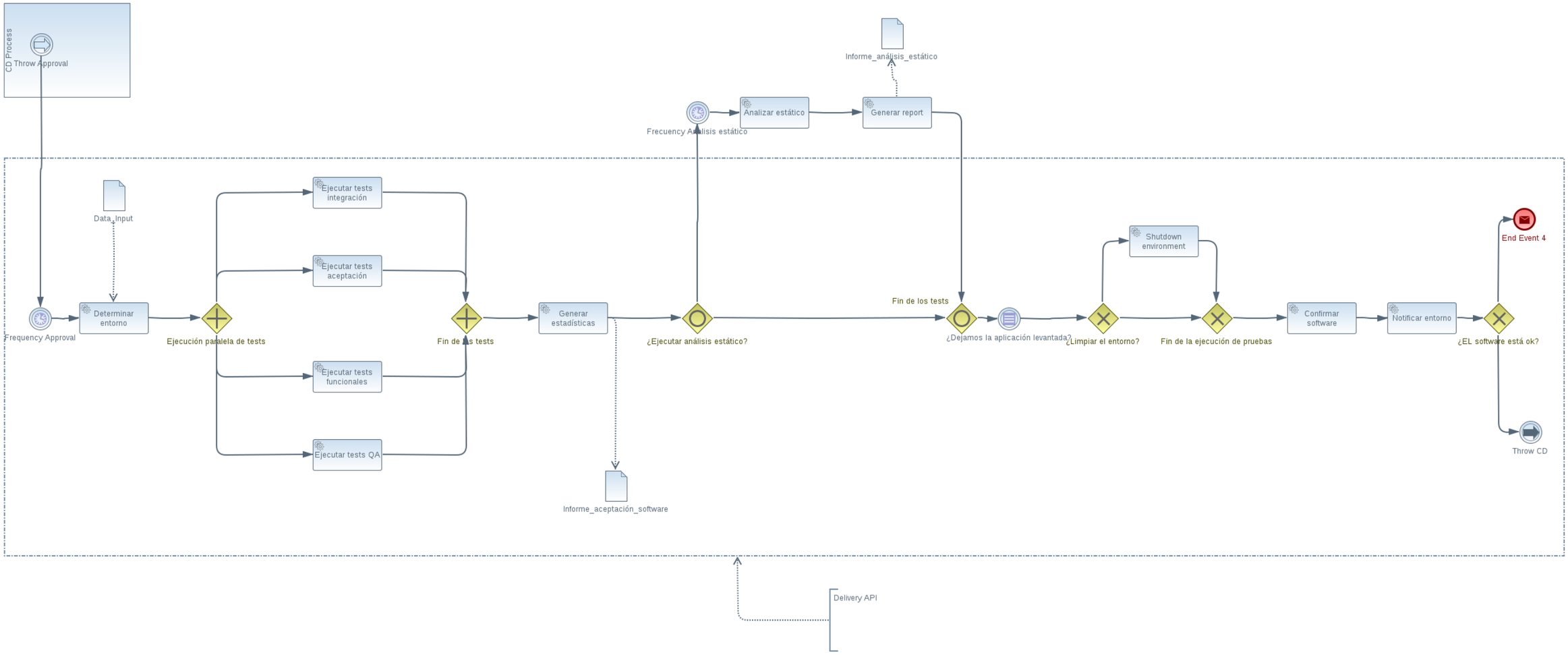
En este proceso modelo las acciones que, a mi criterio, son parte de esta fase de DevOps. Entiendo que puedas pensar que faltan cosas o que incluso no es aquello que tenías en mente. Bien, he intentado dividir un proceso complejo en diferentes subprocesos más comprensibles y simples, lo que provoca que, al observar una proceso aislado, puedas percibir que faltan cosas, que el diseño no termina de cobrar del todo sentido o que hubieses añadido, en este modelo, tareas que has identificado en otros procesos. Es perfectamente entendible, ya que cada uno puede ver y diseñar el mismo sistema de manera diferente, lo importante es dotar al diseño de nuestra arquitectura de sentido, sin llegar a complicar los modelos que la conforman. Esta es la razón por la que he segmentado el proceso DevOps y por la que recomiendo la división de los procesos de negocio una vez que alcanzan cierto tamaño.

Ahora te quiero hablar de dos detalles. Por un lado, hago un diseño en el que el paso del software a un entorno de producción siga siendo bajo confirmación manual, es la parte que puedes ver remarcada como “Confirmación subida a Producción”, es algo que no encaja bien con la filosofía DevOps, pero si trabajas en una gran empresa y más aún en entorno bancario, eres consciente de que no deja de ser una realidad incluso para entornos previos al mencionado. Si investigas sobre DevOps y lo que implica, puedes pensar que este modelo está mal, que debemos desplegar en producción de manera automática, y te doy la razón completamente. Ahora bien, un aspecto que me gusta tener siempre presente es cómo de factible es la implantación de la arquitectura que estoy diseñando. Si diseño el caso ideal, estaría diseñando un sistema que, en cierto modo, no llegará a ser tal y en caso de que lo sea, llevará consigo un coste elevado. En algunas ocasiones debemos de ser realistas, diseñar arquitecturas más simples y sencillas de implantar.

Por otra parte, intento identificar otro conjunto de tareas completamente automatizables bajo el nombre “Delivery API”. Dentro de este grupo están las acciones que se dispararán cuando termine el proceso de integración continua o cuando un software requiera una promoción de entorno (teniendo en cuenta que, como he comentado, la llegada a producción implicará una confirmación manual por parte del dueño del software y del equipo técnico que lo haya desarrollado).

De nuevo, puedes observar como conecto diferentes procesos de negocio mediante eventos. Unos que provocan el inicio del proceso (Ej. En el panel perteneciente a “CI process”, el evento “Throw CD”) y otros, que al terminar, disparan el siguiente proceso de negocio (Ej. “Throw Approval”)

9. Aceptación del software



En este proceso se modelan las pruebas del software que deben ejecutarse paralelamente. En nuestro caso, pruebas de integración, pruebas de aceptación, pruebas funcionales y pruebas de calidad. Es importante generar las estadísticas oportunas de los diferentes tipos de pruebas y hacerlas visibles al equipo de desarrollo, en ese aspecto es clave el “Informe_aceptación_software”.

Además de esto, debemos tener en cuenta si debemos dejar la aplicación corriendo en el entorno en el que estamos ejecutando las pruebas (a modo de demo, para que podamos mostrar nuestro software funcionando) y notificar el estado del software (resultado de las pruebas realizadas). Esto nos permitirá tener un registro con la información necesaria para valorar si podemos promocionar nuestro software al siguiente entorno que hayamos definido.

Como puedes ver, también indico que debe realizarse un análisis estático completo del código, que por ser un proceso bastante costoso en cuanto a tiempo y procesamiento, lo ejecutaremos con una frecuencia bastante más reducida que el resto del proceso (Ej. Una vez al día).

De nuevo aparece un grupo “Delivery API” que, como en el caso anterior, son tareas a automatizar, bien sea mediante algún tipo de desarrollo propio o algún producto disponible en el mercado. Mi intención es que este grupo de tareas sean lanzadas una vez que tengamos la aplicación corriendo en un entorno determinado. Esto queda reflejado mediante el evento inicial del proceso “Throw Approval”, evento que como has visto antes se produce en el momento final de la entrega continua.

Si te has fijado en las tareas que incluyo dentro de “Delivery API”, habrás notado que he dejado fuera el análisis estático. Ya que este grupo de acciones estará dentro del flujo global de DevOps y una de nuestras metas es conseguir un tiempo de ciclo lo más reducido posible, recomiendo mantener fuera tareas de este tipo. En este caso, no son críticas a la hora de valorar si el software cumple con los requisitos establecidos, puesto que en la fase de integración continua he modelado un análisis estático incremental, es decir, similar a este pero sólo ejecutado sobre los últimos cambios en el software.

10. Conclusión final

Es clave que, aunque se decida cambiar la herramienta utilizada para cubrir la integración continua, migrar el “Delivery API” a nueva tecnología o incluso sustituirlo por un producto, nuestros modelos BPMN seguirán siendo los mismos. Nuestro nuevo api o nuestro nuevo producto deberá proporcionar la funcionalidad descrita y las personas encargadas de la implementación de ese nuevo API o de la configuración de esa herramienta, deberán cumplir con lo definido en los modelos BPMN expuestos anteriormente.

Te comento esto, ya que este proyecto tiene bastante grado de implantación y en mi caso particular ya se me está presentando esta situación. La necesidad de gestionar varios entornos, manejar los ficheros de infraestructura de aplicación de una manera más adecuada, ofrecer servicios REST/SOAP... hacen que para la evolución del “Delivery API” me plantee un cambio tecnológico. En este proceso de migración, serán clave los modelos tanto de entrega continua como de aceptación del software, ya que es donde tenemos documentada su funcionalidad.

Como puedes intuir, DevOps no es algo que técnicamente sea complejo o tedioso de llevar a cabo. Desde mi parecer, su complejidad reside en el aspecto metodológico del proceso, es decir, cómo se deben ejecutar las tareas, cuándo, dónde hay que dejar las evidencias de las ejecuciones, cuáles son los criterios de paso de un proceso a otro, qué trazabilidad debemos de conservar a lo largo del circuito, qué tareas deben de ser manuales, qué acciones requieren una confirmación especial... son detalles (extrapolables a cualquier otro proceso de negocio) que debemos de tener completamente claros, de otro modo, crearemos un ciclo DevOps (o un sistema de software) que no se adecuará a nuestra organización y que, por tanto, no terminará de tener la aceptación deseada dentro de la misma.

Debido a todas estas razones, es sumamente importante el modelado de DevOps haciendo uso de los lenguajes estándar, en este caso ArchiMate y BPMN. Esto nos permitirá tener una documentación de gran valor, independiente de la tecnología que lo implemente, y entendible por todos los participantes del proyecto. Conseguiremos así, clarificar el sistema desde todos los puntos de vista afectados, de modo que estén conformes con el diseño de aquello que se llevará a cabo.

Si has pasado por mi anterior artículo, habrás observado que el proceso de diseño que he seguido es el contrario En este caso, he comenzado desde un modelo con poca granularidad (*5. DevOps como proceso de negocio*) y he ido progresando a otros con más detalle y complejidad (Ej. *8. Aceptación del software*). En mi opinión, el resultado tiende a ser el mismo.

Decirte que BPMN cuenta con una gran variedad de elementos diferentes, en ese sentido, su curva de aprendizaje es ligeramente mayor a la de otros lenguajes que hayas apredido. No obstante, no es complicado y te animo a que en tu próximo proyecto lo utilices (si aplica, claro está) y compruebes de primera mano el valor y utilidad que te estoy comentando.

Para aquellos que tras estos dos artículos notéis que os ha sabido a poco, que os falta algo más, deciros que a mí también. Por ello, publicaré un último artículo centrado principalmente en DevOps y en cómo llevarlo a cabo partiendo de esta arquitectura que hemos modelado. Considero una buena idea separarlo a fin de no mezclar el diseño de la arquitectura con la implementación de la misma.

Llegados a este punto, alentarte a que completes el círculo con el siguiente artículo. Además, agradecerte que hayas terminado la lectura de éste y que cuando recapitules lo que he intentado mostrarte, le encuentres valor y sentido.