**Metodología Scrum**

Primero que nada definiremos lo que es una metodología ágil se entiende como los métodos de la ingeniería del software basados en el desarrollo iterativo e incremental, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupos autos organizados y multidisciplinarios. Ahora scrum es una metodología ágil y flexible para gestionar el desarrollo de software, cuyo uno de sus principales objetivos es poder maximizar el retorno más bien conocido como ROI. En Scrum se caracteriza por tener entregas o avances parciales del producto final. En este sentido esta metodología esta principalmente orientado a entornos de mucha complejidad. Donde se prioriza siempre la innovación, la competencia, flexibilidad y alta complejidad.

El proceso de esta en un proyecto se genera en periodos temporales, cortos y fijos. Cada iteración tiene que entregar un resultado completo. Existen formas de planificar cada iteración primero parte mediante la selección de requisitos más prioritarios esta actividad por lo general toma cuatro horas como máximo. Para luego planificar la iteración el equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos que se habían comprometidos.

La manera en que se ejecuta la iteración es mediante una pequeña reunión todos los días de quince minutos como máximo en esta reunión cada miembro del equipo responde a tres preguntas básicas las cuales son: ¿Qué he hecho desde la última reunión de sincronización?-¿Qué voy a hacer a partir de este momento?-¿Qué impedimentos tengo o voy a tener? Donde también dentro de eseequipo presentan roles bien definido tales como Product Owner, Scrum master, equipos de desarrollo y otros roles menos claves que se conocen como roles auxiliares.

Para concluir Scrum es una metodologíaque permite cumplir fácilmente alas expectativas generadas por el cliente, tiene una alta capacidad de ser flexible al cambio, permite entregar una mayor calidad del software , permite tener predicciones de tiempo lo que se logra mediante los llamados sprint que permite estimar cuantos tiempos se demorara el equipo y por ultimo permite reducir radicalmente el riesgo ya que mediante las diferentes reuniones se pueden anticipar las diferentes coyunturas.

Metodología cascada

Ciclo de vida clásico o en cascada este enfoque del desarrollo de [software](http://jummp.wordpress.com/2011/12/03/que-es-el-software/) ha sido el más utilizado y en la actualidad, pese a la aparición de metodologías [ágiles](http://jummp.wordpress.com/agile/), sigue siendo la solución predominante, si bien, en cada organización o en cada de [proyecto](http://jummp.wordpress.com/2011/11/23/desarrollo-de-software-que-es-un-proyecto/) se puede llevar a cabo con ciertas variantes.

Llas fases en que se divide el [ciclo de vida clásico](http://jummp.wordpress.com/2011/12/03/que-es-el-ciclo-de-vida-del-software/), pueden ser diferentes. Pero segun las que estableció Roger Pressman (no obstante para la descripción del alcance de las mismas voy a basarme principalmente en Métrica V.3, pero adaptándome a esta clasificación), si bien otras muchas alternativas podrían ser válidas:

- Análisis: Esta fase comprendería desde la posible obtención de unos objetivos o requisitos iniciales para determinar la viabilidad del sistema y escrutar las distintas alternativas de solución, pasando por la elaboración del catálogo de requisitos, hasta la realización de casos de uso, prototipado de pantallas e informes, como una primera especificación del plan de pruebas.

- Diseño: El análisis describe el sistema sin entrar en características propias de la implementación, es en esta fase donde se adapta ese análisis generalista a la solución concreta que se quiere llevar a cabo, definiéndose la arquitectura general del sistema de información, su división en subsistemas de diseño, el modelo de datos lógico, el modelo de clases (en el caso de un diseño orientado a objetos), la especificación detallada del plan de pruebas, etc…

- Codificación: En esta fase se realiza la construcción del sistema de información y las pruebas relacionadas con dicho proceso, como son las unitarias, integración y de sistema, así como otras actividades propias de las etapas finales de un desarrollo como es la realización de la carga inicial de datos (si bien en muchos casos se deja esto para cuando el producto está en producción) y/o la construcción del procedimiento de migración.

- Pruebas: En esta etapa se realizaría la instalación del sistema en un entorno de pruebas lo más parecido posible al de producción (entorno de preproducción) donde se realizarían las pruebas de implantación (que verifican principalmente aspectos no funcionales) y las de aceptación, donde los usuarios validan que el sistema hace lo que realmente esperaban (sin que se deba olvidar que los límites los establecen los modelos realizados previamente y que han debido ser validados). Por último se realizaría la implantación del sistema en el entorno de producción.

- Mantenimiento: Una vez que el sistema se encuentra en producción, se realizarán sobre el mismo diversas tareas de mantenimiento, que en función de su naturaleza se clasifican en correctivos, evolutivos, adaptativos y perfectivos. Estas tareas de mantenimiento serán consecuencia de incidencias y peticiones reportadas por los usuarios y los directores usuarios.

En el ciclo de vida clásico en función de las modificaciones y/o correcciones que se realizan en una etapa será necesario la vuelta a fases previas para hacer coherente el proceso de desarrollo y los modelos.

El principal inconveniente que se le ha achacado siempre a este tipo de ciclo de vida es que los usuarios tardan demasiado en ver los resultados, lo que hace que el tiempo transcurrido desde que se define el sistema hasta que está disponible sea lo suficientemente amplio como para que hayan ocurrido muchas cosas: desde que no estén la mayoría de las personas que participaron en la especificación, como cambios en los procesos, cambios de criterio, etc…, lo que provoca a hacer replantemiento de los requisitos en etapas más tardías del desarrollo con el coste que eso conlleva (un refinamiento de los requisitos es razonable siempre y cuando no se superen unos límites) y a que muy probablemente el sistema finalmente disponible esté alejado de lo que realmente quiere el conjunto de usuarios en estos momentos que es distinto a lo que querían hace meses y a lo que querrán dentro de otros tantos.

Otro inconveniente ligado con el anterior es que al tratarse el sistema como un todo, los modelos generados (catálogo de requisitos, casos de uso, etc…) serán lo suficientemente grandes como para no poder ser revisados y comprendidos en toda su magnitud, sobre todo por personal no informático. Esto crea una incertidumbre sobre los requisitos del sistema que más adelante traerá problemas.

No todo es malo en este modelo, ya que describe un procedimiento racional y ordenado de desarrollo de software, la clave para su éxito o su fracaso es como se gobierne el mismo y las circunstancias que rodeen al proyecto en el momento de su ejecución. Además, existen variantes que ofrecen una mayor flexibilidad al mismo y que permite reducir sus riesgos.