Readiness in software engineering

Carlos Cabrera Moral

Reflexión sobre el concepto de Readiness y el Requirements Maturation Flow (RMF)

En el desarrollo de software moderno es habitual asumir que cuando antes se empiece a programar, mejor. La presión por mostrar avances funcionales y cumplir plazos empuja a los equipos a "ponerse con el código" lo antes posible. Sin embargo, muchos proyectos que siguen esta lógica terminan rehaciendo trabajo y acumulando retrasos y frustración.

El concepto de Readiness, desarrollado por Max Guernsey III y Luniel de Beer, parte de la siguiente idea: muchos proyectos no fallan por falta de talento técnico, sino porque empiezan a implementarse antes de estar realmente preparados.

Para abordar este problema proponen el Requirements Maturation Flow (RMF), un marco de trabajo que introduce estructura en el proyecto previo a la implementación. No pretende sustituir metodologías ágiles como Scrum, sino reforzarlas desde un punto crítico que suele descuidarse: la maduración real de los requisitos.

Empezar demasiado pronto

En muchos equipos ágiles se realizan sesiones de refinamiento del backlog donde se revisan historias de usuario, se aclaran dudas rápidas y se asignan estimaciones. Sobre el papel parece suficiente. Sin embargo, en la práctica, estos refinamientos suelen ser superficiales y genéricos.

Un requisito puede parecer claro cuando se formula de manera breve. Por ejemplo:

> *Si el tipo de cambio introducido por el usuario supera el rango diario permitido, mostrar un mensaje de advertencia.*

A primera vista parece una historia sencilla. Pero cuando el equipo empieza a hacer preguntas surgen múltiples ambigüedades: ¿Cómo se define ese rango? ¿De dónde proviene el dato? ¿El mensaje bloquea la operación o solo informa?

Lo que parecía simple deja de serlo. Y si el equipo no realiza estas preguntas antes de implementar, es muy probable que entregue algo técnicamente correcto, pero funcionalmente incorrecto o diferente de lo esperado.

Este tipo de desalineación es lo que RMF intenta evitar.

RMF1: El entendimiento compartido como punto de partida

El primer componente del Requirements Maturation Flow es garantizar un entendimiento compartido entre todas las partes implicadas. No basta con que el requisito esté escrito, es necesario que producto, stakeholders y desarrolladores tengan la misma interpretación del resultado esperado.

Este proceso implica dedicar tiempo específico a analizar un único ítem, plantear preguntas profundas y eliminar ambigüedades. La clave es no asumir que existe alineación solo porque nadie expresa dudas.

RMF introduce la idea de trabajo de maduración: actividades cuyo objetivo no es programar, sino reparar el requisito hasta que el equipo tenga la seguridad suficiente para implementarlo. Esta practica reduce significativamente los malentendidos y las dudas.

RMF2: Definición de Done

En muchos equipos existe una definición de "Done" genérica que aplica a todas las tareas. RMF propone algo distinto: cada elemento del backlog debería tener una definición de Done adaptada a su naturaleza.

No todos los trabajos son iguales. Un cambio en la base de datos, una modificación en el a interfaz o una integración con un sistema externo requieren condiciones distintas para considerarse terminados.

Al definir explícitamente qué significa "hecho" para ese ítem concreto, se crea un acuerdo formal entre producto y desarrollo. Este acuerdo reduce discusiones al final del sprint y mejora la precisión de las estimaciones, ya que el equipo entiende con claridad qué condiciones deben cumplirse para cerrar la tarea.

RMF3: Definición de Ready

Si la definición de Done protege el final del proceso, la definición de Ready protege el inicio.

Un ítem no debería entrar en un sprint de implementación hasta que ambas partes acuerden que está realmente preparado. Esto implica que las especificaciones estén maduras, que las dependencias estén identificadas y que no existan bloqueos evidentes.

La definición de Ready suele contemplar dos dimensiones:

- Que el requisito esté suficientemente definido desde el punto de vista funcional.
- Que el entorno técnico esté preparado para soportar su implementación.

Esta práctica evita comenzar a desarrollar algo que todavía depende de aclaraciones externas o cambios arquitectónicos no resueltos.

¿Retrasa esto el desarrollo?

Puede parecer que dedicar tiempo a madurar requisitos ralentiza el proyecto. Sin embargo, la propuesta de RMF sostiene que no retrasa el avance, sino el error. Corregir antes de programar es significativamente más barato que corregir después.

Además, trabajar con requisitos claros suele aumentar la motivación del equipo. Programar en un entorno de incertidumbre constante genera frustración, hacerlo sobre bases bien definidas genera confianza y estabilidad.