Priorización de funcionalidades

Contenido

Resumen

- **Versión:** v01r01
- **Fecha publicación:** 30 de Julio de 2021

Histórico de cambios

Los cambios en la documentación de apoyo vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución.

Versión	v01r01	**Fecha publicación**	30 de Julio de 2021
Alcance			
• Presentación de varios métodos para priorizar funcionalidades y grandes iniciativas (épicas o mejoras)			

Introducción

¿Qué funcionalidad debería ser la siguiente a incluir en mi aplicativo?

Es una pregunta que las/os Dueñas/os de Producto (o **Product Owners**) suelen encontrarse y cuya respuesta, a veces, no es nada trivial. Para ayudar en esta tarea, en esta página se exponen varias técnicas para abordar la priorización de funcionalidades de una manera sistemática.

El uso de una técnica de priorización u otra depende de las necesidades del proyecto así como de las preferencias de los/as Product Owners. En ocasiones, incluso, es posible que sea conveniente utilizar una combinación de varias técnicas.

Técnicas de priorización

Priorización por Objetivos

Usar en caso de querer priorizar un subconjunto de objetivos específicos frente al resto.

Consiste en priorizar las funcionalidades siguiendo una visión enfocada a objetivos siguiendo los siguientes pasos:

1. Crear una lista de objetivos a conseguir por tu aplicativo. En la descripción del objetivo es importante centrarse en los qué se quiere conseguir y no en cómo va a conseguirse.
2. Priorizar dicha lista de objetivos
3. Seleccionar 3 de esos objetivos para focalizar esfuerzos
4. Por cada objetivo, elabora la lista de funcionalidades que pueden acercarte a dicho objetivo
5. Asignar mayor prioridad a aquellas funcionalidades de las que se espera tener un mayor rendimiento. Este paso es interesante compatibilizarlo con el modelo RICE más abajo descrito.

Esta priorización comparte una filosofía similar a la del Taller de Mapas de Impacto, especialmente utilizado en el lanzamiento de proyectos.

Encuesta a Usuarios con el Modelo KANO

Usar en caso de querer priorizar en base al feedback directo de los usuarios de nuestro aplicativo.

Consiste en la **categorización** de las diferentes funcionalidades como requeridas, unidimensionales, atractivas, indiferentes o inversas, a través del **uso de cuestionarios** enviados a nuestros usuarios. Esta categorización nos permite hacer una priorización inmediata de nuestras funcionalidades.

Para ello, el modelo Kano propone un cuestionario estandarizado, en el que los participantes deben responder dos preguntas para cada funcionalidad del producto: una está formulada de manera positiva (funcional) y la otra está formulada de manera negativa (disfuncional).

De la combinación de las respuesta a la pregunta funcional y disfuncional de cada usuario para cada funcionalidad, puede inferirse la siguiente clasificación:

Funcional		Disfuncional	Categoría	Prioridad
Cuento con ello	+	No me gustaría	**REQUERIDA** / aquellas funcionalidades que se *dan por sentado* cuando se cumplen, pero que generan *insatisfacción si no se cumplen*.	MUY ALTA
Me gustaría	+	No me gustaría	**UNIDIMENSIONAL** / aquellas funcionalidades que *satisfacen al usuario cuando se cumplen* y generan *insatisfacción si no se cumplen*.	ALTA
Me gustaría	+	Soy neutral	**ATRACTIVA** / aquellas funcionalidades que generan *satisfacción cuando se cumplen*, pero *no causan insatisfacción* cuando no se cumplen.	MEDIA
Soy neutral	+	Soy neutral	**INDIFERENTE** / funcionalidades que no generan satisfacción ni insatisfacción si se incluyen o no se incluyen	BAJA

No me gustaría	+	Cuento con ello	**INVERSA** / cuando más es menos (p.ej. hay clientes que prefieren modelos básicos de funcionalidad)	MUY BAJA /DESECHAR

Así pues, un ejemplo de cuestionario para una funcionalidad sería:

EJEMPLO 1	**Me gustaría**	**Cuento con ello**	**Soy neutral**	**Puedo tolerarlo**	**No me gustaría**
Pregunta funcional: ¿Cómo te sentirías si el producto tuviera...?					
Pregunta disfuncional: ¿Cómo te sentirías si el producto no tuviera...?					

> Nota 1: Las preguntas anteriores podrían sustituirse por "¿Cómo te sentirías si hubiese más/menos de...?
>
> Nota 2: Suelen descartarse aquellas respuestas ilógicas (p.ej. tanto la pregunta funcional como la disfuncional están respondidas con un "me gustaría")

Feedback de usuarios con la técnica "Compra una Funcionalidad"

Usar en caso de querer priorizar en base al feedback directo de los usuarios de nuestro aplicativo.

Es una técnica de priorización en la que el equipo de producto (Product Owners, Equipo de desarrollo,..) interacciona directamente con los usuarios para descubrir cuáles son las funcionalidades mas valoradas por nuestros usuarios.

Para llevarla a cabo, se siguen los siguientes pasos:

1. Preparar un **lista de funcionalidades**, que es recomendable limitar a un máximo de 30 funcionalidades
2. **Ponerle precio a cada funcionalidad** en función del esfuerzo/complejidad que el equipo de desarrollo prevea. Los precios a utilizar son: **500** monedas, **1.000** monedas, **2.000** monedas, **3.000** monedas, **5.000** monedas, **8.000** monedas, **13.000** monedas y **20.000** monedas.
3. Organizar la sesión con max. 6 usuarios (aunque también puede hacerse uno a uno). Asignar un presupuesto total al grupo del 40% del coste de todas las funcionalidades.
4. Invitar a los usuarios a que lleguen a consensuar qué funcionalidades comprar con el dinero asignado al grupo (en caso de hacerlo de manera individual, usuario a usuario, el proceso se simplifica).
5. Anotar todos los comentarios que se hagan acerca de las diferentes funcionalidades. Cuando se hace esta dinámica de manera grupal, las conversaciones relacionadas pueden ser de gran valor.

Aunque para esta dinámica puede utilizarse un formulario online u otra opción interactiva, también podría utilizarse la siguiente plantilla:

Descargar Excel de "Comprar una funcionalidad"

Priorización interna del MVP en lanzamientos - Método MoSCoW

Usar en caso de querer configurar un MVP (Produto Mínimo Viable) en el lanzamiento de un proyecto, tanto de nueva creación como en migraciones tecnológicas.

MoSCoW (de "**M**ust have", "**S**hould have", "**C**ould have" y "**W**on´t have") es una técnica de priorización especialmente recomendada para proyectos con deadlines fijas y alcance variable. Es especialmente recomendable para la definición del alcance de un MVP (Producto Mínimo Viable). Esta técnica consiste en la clasificación de las funcionalidades proyectadas en 4 categorías, que por orden de prioridad son:

- **Must have** - Aquellas funcionalidades que sí o sí **DEBEN** estar contenidas en la versión ya que de lo contrario:

  (a) no tiene sentido liberar la versión en la fecha proyectada

  (b) No sería legal liberar versión

  (c) No sería seguro liberar versión

  (d) No se puede entregar una solución viable sin ellas.

  En una versión con tiempo fijo, la metodología DSDM recomienda no tener *más de un 60% de este tipo de funcionalidades*, ya que de lo contrario comprometería un *riesgo para el proyecto* (excepto cuando: hay confianza plena en las estimaciones, no hay dependencias externas o desconocidas, el equipo conoce perfectamente la solución,...)

- **Should have** - Aquellas funcionalidades que DEBERÍAN estar contenidas en la versión, esto es:

  (a) son importantes pero no vitales

  (b) puede que sea doloroso dejarlas fuera, pero si se dejasen fuera la solución aún sería viable

  (c) es posible que sea necesario un workaround temporal en caso de que se dejen fuera (p.ej. gestión de expectativas, ineficiencias, ...)

- **Could have** - Aquellas funcionalidades que PODRÍAN incluirse, ya que:

  (a) son deseables pero menos importantes

  (b) tienen menor impacto si se dejan fuera

  Estas funcionalidades son las que conforman la contingencia principal del proyecto. La metodología DSDM recomienda que *al menos el 20%* de las funcionalidades sean de esta categoría

- **Won´t have** - Aquellas funcionalidades que de antemano se sabe que no entraran en la versión planificada.

Antes de realizar una sesión de priorización, es importante alinear el criterio para generar consenso para asignar una categoría a cada funcionalidad (lo que decida finalmente el Product Owner, decisión por mayoría, decisión por la norma,...).

En una variante de este método, el alcance funcional puede combinarse con el tecnológico.

Modelo RICE

Usar en caso de ser capaz de estimar: el *número de usuarios impactados* aproximados, el *impacto de la funcionalidad* sobre los mismos, la *confianza* que se tiene sobre la eficacia y efectividad de la funcionalidad y el *esfuerzo* aproximado para desarrollarla.

El modelo RICE asocia una puntuación a la prioridad de cada funcionalidad en base a 4 factores: **REACH** o usuarios impactados en un intervalo de tiempo determinado, **IMPACTO** que tendrá la funcionalidad sobre cada usuario, **CONFIANZA** que tenemos respecto a nuestras estimaciones y **EFFORT** o esfuerzo que requerirá del equipo de desarrollo.

¿Cómo calcularíamos la puntuación RICE? Primero deberíamos estimar nuestros 4 factores:

- ***REACH - ¿Qué número de usuarios se verán impactados?*** La precisión en el número exacto de usuarios no es tan relevante como sí lo es utilizar siempre el mismo criterio temporal (p.ej. no comparar una funcionalidad que impacta a 100 usuarios/semana con una que impacta a 100 usuarios/año)
- ***IMPACTO - ¿Cuánto impactará a cada persona?*** MASIVO=3, MUCHO=2, MEDIO=1, BAJO=0.5 y MÍNIMO=0.25
- ***CONFIANZA - ¿Qué confianza tenemos en la relevancia de la función?*** ALTA=100%, MEDIA=80%, BAJA=50%
- ***ESFUERZO - ¿Cuánto Meses-Persona creemos que tomará desde definición hasta puesta en producción?*** Importante utilizar números enteros para no perderse con la exactitud en las estimaciones

Finalmente, el cálculo de la prioridad se hará con la siguiente fórmula:

$$\textbf{Puntuación RICE} = \frac{Reach \cdot Impacto \cdot Confianza}{Esfuerzo}$$

En la siguiente hoja de cálculo Excel puede verse una plantilla de este modelo

Descargar plantilla Modelo RICE

Método WSJF (Weighted Shortest Job First)

Usar en suites de aplicativos (o varios equipos ágiles) donde sea interesante hacer una priorización de funcionalidades de alto nivel. Además, es necesario poder puntuar de manera correlativa entre las funcionalidades el *Valor de Negocio*, la *Urgencia*, el *Impulso de Oportunidades*, la *Reducción de Riesgos* y el *Esfuerzo*.

Es un método común de priorización de funcionalidades en portfolios/programas comerciales.

El método "Primero el trabajo ponderado más corto", prioriza aquellos items del backlog con *mayor coste de retraso* que tengan *menor esfuerzo*.

El método aquí presentado es una adaptación del original, definiéndose el coste de retraso como:

$$\textit{COSTE DE RETRASO} = \textit{Valor de Negocio} + \textit{Urgencia} + \textit{Impulso de Oportunidades} + \textit{Reducción de RIESGOS}$$

Finamente, para obtener el "**trabajo ponderado más corto**" (WSJF), se divide el *Coste de Retraso* de la funcionalidad entre el *Esfuerzo* asociada a la misma:

$$WSJF = \frac{\textit{COSTE DE RETRASO}}{\textit{Esfuerzo}}$$

En definitiva, en este método se prioriza un mayor WSJF.

Editar plantilla método WSJF aquí

Matriz importante-urgente (o de Eishenhover)

Usar en caso de buscar un método de priorización de funcionalidades sencillo para aplicativos en mantenimiento.

Consiste en clasificar doblemente cada funcionalidad como "urgente/ no urgente" y como "importante / no importante". Así pues, la recomendación es: HACER lo "*urgente-importante*", PLANIFICAR lo "*no urgente-importante*", ENCOLAR (o delegar) lo "*urgente-no importante*" y DESECHAR lo "*no urgente-no importante*".

Ejemplo de Matriz Eishenhover en el desarrollo de un Teléfono Móvil