Universidad ORT Uruguay
Facultad de Ingeniería

Ingeniería de Software Ágil 2

Guía de “Definición del proceso de ingeniería en el contexto de Kanban”

Clavijo, Tomás (235426)

Barreto, Sofía (258216)

Facello, Bruno (260694)

Docentes: Alvaro Ortas, Alejo Pereira

2023

Índice

Introducción

Este documento reúne la información correspondiente a la definición del proceso de ingeniería en el contexto Kanban.

Presentamos una descripción detallada del proceso de identificación y corrección de bugs. En particular, se explican los pasos seguidos para la realización de pruebas y revisión del código, señalando en cada paso el resultado. Además, se detallan los roles de cada miembro del equipo en el proceso, así como las herramientas y metodologías utilizadas para llevar a cabo las pruebas y la revisión del código.

Hablaremos del origen de Kanban y sus principios que nos han llevado a tomar las decisiones correspondientes.

Finalmente, explicaremos el funcionamiento del tablero elegido y cómo podría mutar a futuro.

Definición del proceso de ingeniería en el contexto de Kanban

Origen de Kanban

La palabra japonesa “kanban” significa “tablero visual”, y se utiliza en el entorno de definición y mejora de procesos desde la década de 1950.
Fue desarrollado y aplicado por primera vez por la empresa Toyota como sistema de programación para la fabricación JIT (Just In Time).
Representa un sistema con un comportamiento “pull”, es decir, la producción se basa en la demanda de los clientes. Busca crear más valor para el cliente sin generar costos extra.
A principios del siglo XXI, se opta por llevar Kanban a sectores más complejos como la informática, el desarrollo de software, entre otros.

Kanban como marco de ingeniería ágil

En un proceso de ingeniería, cuando nos referimos a gestión, hacemos referencias a las actividades que comprende el Ciclo de Deming, comúnmente conocido como su acrónimo “PCDA” o también “plan, do, check, act”.
Kanban sigue el modelo de gestión de Lean Manufacturing, que tiene como objetivo minimizar las pérdidas y maximizar el valor añadido al cliente.
Determina seis prácticas que se deben dominar para una implementación exitosa:

- Visualizar el flujo de trabajo
- Limitar el trabajo en curso (“work in progress”, WIP)
- Gestionar el flujo
- Explicitar las políticas y procedimientos
- Aplicar bucles de retroalimentación
- Mejorar y evolucionar

Comenzando el proyecto, a grandes rasgos sabemos que se desarrollaran tareas de testing, con el objetivo de encontrar bugs en el código de terceros y tareas de desarrollo, para implementar nuevas funcionalidades, por lo que definimos los siguientes ***roles de equipo***:

Desarrollador: Es un profesional de la informática que se dedica a diseñar, programar y mantener aplicaciones, programas y sistemas de software. En general, su trabajo consiste en escribir código para que los programas y sistemas funcionen correctamente y de manera eficiente.

Tester: Un tester, también conocido como un analista de pruebas o QA (Quality Assurance), es una persona que se dedica a probar el software para asegurarse de que cumpla con los requisitos funcionales y no funcionales. Su trabajo consiste en encontrar errores o defectos en el software y documentarlos para que puedan ser corregidos antes del lanzamiento del producto.

Product Owner: Es una persona responsable de definir y priorizar los requisitos y funcionalidades de un producto de software. Es el encargado de asegurarse de que el equipo de desarrollo esté trabajando en las características más valiosas y relevantes para el cliente y el usuario final, y de que el producto cumpla con las necesidades del mercado.

Miembro	Roles
Sofía Barreto	Developer, Tester, Product Owner
Tomás Clavijo	Developer, Tester
Bruno Facello	Developer, Tester

Durante el proceso de **Identificación de bugs**, utilizamos diferentes métodos para identificar posibles errores o problemas en el funcionamiento. En primer lugar, realizamos pruebas exploratorias, donde evaluamos la aplicación de manera libre y sin seguir un guión específico para encontrar posibles problemas o errores.

Pruebas exploratorias:

- Resultado: Identificación de posibles errores en la funcionalidad de la aplicación.
- Rol: Testers.
- Momento: Durante todo el proceso de Identificación de bugs.

A través de estas pruebas, pudimos detectar errores en la funcionalidad de la aplicación, como problemas de navegación y errores de carga de datos.
Sin embargo, no fueron las únicas, también realizamos pruebas de usabilidad para evaluar la experiencia del usuario al interactuar con la aplicación. En estas pruebas, nos enfocamos en la facilidad de uso, la accesibilidad y la eficiencia de la aplicación.

Pruebas de usabilidad:

- Resultado: Identificación de problemas de usabilidad, accesibilidad y diseño de la aplicación que podrían afectar la experiencia del usuario.
- Rol: Testers.
- Momento: Después de realizar las pruebas exploratorias.

Pudimos identificar problemas de usabilidad, de accesibilidad, así como también en el diseño, que podrían afectar la experiencia del usuario. En cada paso, definimos claramente los roles de cada miembro del equipo en la identificación y corrección de los posibles errores o problemas encontrados.

Continuando con las **nuevas funcionalidades** así como también la solución de los **issues** presentes, decidimos que previo al desarrollo se deben definir de manera clara los requerimientos. Para esto, a los roles definidos anteriormente le agregamos uno más:

Ingeniero de Requerimientos: Es el encargado de analizar, formular y priorizar los requisitos de un proyecto. Identifica restricciones y servicios. Requisitos bien definidos aseguran una menor probabilidad de fallos en el desarrollo del proyecto.

Continuando con el proceso definimos que la forma más práctica para el IR de definir los requerimientos es a través del uso de Users Stories (historias de usuario).
Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final. Su propósito es determinar cómo proporcionará valor al cliente una determinada función.
La misma comenzará desde el Backlog y se irá ejecutando según el flujo determinado en Kanban.
La plantilla a utilizar seguirá la siguiente estructura: Como [Perfil] [Quiero] [Para], buscando determinar la siguiente información:

- Como: ¿Para quién se desarrolla? El perfil de la persona, en detalle, se debe sentir empatía con la misma.
- Quiero: La intención, ¿qué quiero lograr? Debe ser independiente de la implementación o cualquier otro detalle relacionado.
- Para: ¿Cuál es el beneficio que se busca? ¿Cuál es el problema a resolver?

Como se mencionó, estas historias de usuario estarán realizadas una vez que atraviesen todo el flujo llegando a Done. Sin embargo, no todas pueden ser atendidas al mismo tiempo, porque el **WIP es limitado.**

WIP por sus siglas (“Work In Progress”), es la cantidad de tareas que un equipo está trabajando en el momento. La **limitación del WIP** permite crear un flujo de trabajo ininterrumpido, restringe la cantidad máxima de elementos de trabajo en las diferentes etapas. Al aplicar estos límites se posibilita la identificación de cuellos de botella, y así se permite trabajar en ellos antes que se transformen en un bloqueo.
En nuestro caso específico será necesario limitar el WIP a una capacidad menor o igual a tres, siendo que el equipo está formado por únicamente tres integrantes que atenderán una tarea a la vez.

Previo al desarrollo en sí de la aplicación se pueden implementar casos de prueba.
BDD (Behavior Driven Development), denominado en español como desarrollo guiado por comportamiento. Es un proceso de software ágil que busca la colaboración y entendimiento entre desarrolladores, gestores de proyecto y equipo de negocio. Es decir, es el camino para tomar antes de la fase de testing de un proyecto.
Lo llevamos a cabo porque nos permite a todos los implicados entender el proceso desarrollado y el contenido del código fuente. Se especifican los requerimientos como historias de usuario que deberán tener criterios de aceptación:

- Feature: Describe la funcionalidad a desarrollar
- Scenario: Características que se dan para lograr la funcionalidad.
- Given: Predicciones para que se puedan ejecutar las distintas acciones.
- When: Condiciones de las acciones a ejecutar
- Then: Resultado de las acciones ejecutadas

TDD (Test Driven Development), se orienta según los resultados de los casos de prueba definidos por los desarrolladores. Su estructura cíclica garantiza que el código se transmita al sistema productivo únicamente cuando se hayan cumplido todos los requisitos del software.

- Fase roja: Se escribe un test que contenga componentes que aún no hayan sido implementados: una prueba que falle.
- Fase verde: Se programa la solución. Es muy importante redactar únicamente la cantidad de código que sea necesaria, de forma que el test quede marcado en verde.

- Refactoring: Se perfecciona el código, se busca que sea comprensible.

Estas tareas las podrían desarrollar tanto developers como testers, indistintamente; ya que como se mencionó en TDD, por ejemplo, se realizan casos de prueba pero también se codifica con el objetivo de que esas pruebas pasen.
Para el caso particular de las pruebas TDD utilizaremos .Net y el lenguaje C#. BDD se podría implementar utilizando Gherkin, obteniendo casos de prueba.

Ahora sí, finalizado lo mencionado anteriormente, se puede proceder con la implementación de la aplicación. En este caso se define comenzar por el Frontend para continuar hacia el Backend, donde los lenguajes a utilizar serán Angular, Js, C# y .Net. Este trabajo será responsabilidad exclusivamente de los desarrolladores, debiendo participar en todo el proceso. En equipos más grandes se podría llegar a dividir según roles, como por ejemplo, UX/UI Designer.

Por último, repitiendo el proceso que mencionamos al comienzo para identificar bugs, asignándole nuevamente la tarea a los testers, se podría continuar testeando la aplicación, obteniendo el código con los bugs que se presentan, y pasando a la etapa de Refactoring, donde los developers en conjunto con los testers podrán corregir, mejorar, aplicar diversas técnicas que no modifiquen el código pero terminen con los “code smells” que se presenten.

Visto de manera genérica podría quedar algo así:

Qué	Quién	Cómo	Qué obtengo
RD	IR	US	PB
TCI	Dev & Test	Gherkin, C#, .NET	CP
FI & BI	Dev	C#, .NET, Angular	Código
T	Test	PE, PU	Código + Bugs
R	Dev & Test	C#, .NET, Patrones	Código

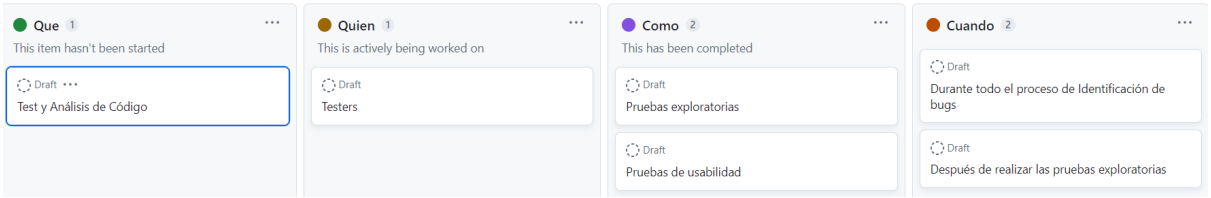
Tablero

En el transcurso del proyecto (como se muestra en la documentación correspondiente), si bien hemos mantenido el uso de Sustentable Kanban, se han ido agregando más columnas que permitan identificar de mejor manera las nuevas necesidades que se van presentando.

1) Tablero primer entrega:

Backlog	Test	A.C	Done

Tablero correspondiente al proceso de ingeniería:



2) Tablero segunda entrega:

Backlog	Investigación de Bug	TDD	Review	Done

3) Tablero tercera entrega:

Sprint Backlog	Requirement definition	Test cases implementation	Frontend implementation	Backend implementation	Testing	Refactoring	Done