DIVISIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

CARRERA

• DESARROLLO DE SOFTWARE MULTIPLATAFORMA•

ACT 3.1 PSP | ESTÁNDARES Y MÉTRICAS

PROFESORA: RUTH BETSAIDA MARTINEZ DOMINGUEZ

FECHA: 21 DE NOVIEMBRE DEL 2024

CUAT. - GRUPO: 4B-DSM

3ER PARCIAL

EDUARDO ALBERTO FLORES HERRERA

UTM UNIVERSIDAD TECNOLÓGICA METROPOLITANA

Índice

Introducción

El Personal Software Process (PSP) es una metodología de desarrollo personal orientada a mejorar la calidad del software y la productividad del desarrollador individual. Fue desarrollada por Watts Humphrey con el objetivo de proporcionar a los ingenieros de software las herramientas y técnicas necesarias para gestionar su propio trabajo de manera más efectiva. Al aplicar el PSP, los desarrolladores pueden seguir un enfoque estructurado que les permite medir y analizar su desempeño, identificando áreas de mejora a través de métricas cuantificables. Este enfoque les ayuda no solo a producir software de mejor calidad, sino también a cumplir con los plazos establecidos y a minimizar los defectos en el producto final.

El PSP se divide en varios niveles que van desde la planificación y el diseño hasta la codificación, compilación, pruebas y análisis retrospectivo. A través de estos niveles, se busca introducir mejoras continuas en el proceso de desarrollo, promoviendo una mentalidad orientada hacia la calidad y la eficiencia. Además, el PSP enfatiza la importancia de la documentación detallada, el registro de tiempos y defectos, y la revisión constante, para que los desarrolladores puedan aprender de sus errores y mejorar iterativamente.

Este documento detalla la aplicación del PSP a un proyecto tutorizado enfocado en la gestión del personal de un hotel. A lo largo de este proyecto, se implementaron prácticas del PSP para mejorar la planificación, la calidad del código y la gestión del tiempo. Se describirán los procedimientos utilizados y cómo estos contribuyeron al desarrollo eficiente del sistema, además de reflexionar sobre los beneficios obtenidos y los desafíos encontrados durante la implementación del PSP en un entorno de desarrollo real.

Entendiendo el PSP

El PSP es una metodología que divide el proceso de desarrollo de software en diferentes fases, cada una con un enfoque específico para mejorar la calidad del producto y la eficiencia del desarrollador. Las fases principales del PSP incluyen:

- **Planificación**: Estimar el tiempo necesario para completar el proyecto y definir los objetivos.
- **Diseño**: Crear una arquitectura que soporte los requerimientos.
- **Codificación**: Implementar el diseño utilizando buenas prácticas de programación.
- **Compilación**: Identificar errores sintácticos y de integración.
- **Prueba**: Detectar defectos en el software para garantizar su correcto funcionamiento.
- **Análisis retrospectivo**: Reflexionar sobre las fases anteriores y realizar ajustes futuros.

Identificación de procedimientos del PSP

Para aplicar los elementos del PSP a mi flujo de trabajo, identifiqué los siguientes procedimientos clave:

- **Registro del Tiempo**: Medí el tiempo invertido en cada fase del desarrollo, utilizando un cronómetro y anotando cada periodo de trabajo en una hoja de cálculo. Esto me ayudó a tener una visión más clara sobre mi productividad y eficiencia. Por ejemplo, dediqué 15 horas a la planificación y diseño del módulo de gestión de empleados, mientras que la fase de codificación tomó 27 horas.
- **Registro de Defectos**: Documenté cada defecto encontrado durante las fases de compilación y prueba, especificando la fase en la que ocurrió, la naturaleza del error, el tiempo invertido en solucionarlo y el impacto que

tuvo. Por ejemplo, durante la implementación del módulo de autenticación, encontré un defecto relacionado con la verificación de roles, que me tomó 5 horas corregir.

- **Medición del Tamaño del Proyecto**: Utilicé métricas como el número de líneas de código (LOC) y el número de funcionalidades para evaluar el tamaño del proyecto. El módulo de gestión de empleados, por ejemplo, consistió en aproximadamente 1577 líneas de código y cubrió funcionalidades como el CRUD de empleados y la asignación de roles.
- **Medición de la Productividad**: Calculé la productividad utilizando el número de líneas de código escritas por hora y el tiempo requerido para solucionar defectos. Durante la fase de desarrollo, logré una productividad promedio de 30 líneas de código por hora, y dediqué un promedio de 2 horas a la corrección de cada defecto identificado.

Aplicación de procedimientos al proyecto tutorizado

Planeación

Durante la fase de planeación, se realizó un análisis detallado de los requisitos del sistema de gestión del personal del hotel. Esta fase incluyó la aclaración de dudas y ambigüedades con respecto a las funcionalidades esperadas, tales como el manejo de roles, autenticación y la gestión de check-in/check-out del personal. Además, se estimó el tiempo necesario para el desarrollo de cada módulo, utilizando técnicas basadas en la experiencia previa y comparando con proyectos similares. Las estimaciones iniciales incluyeron 15 horas para el diseño, 25 horas para la codificación y 20 horas para pruebas.

Actividad	Estimación Inicial (horas)	Tiempo Real (horas)	Diferencia (horas)
Análisis de Requisitos	8	9	+1
Diseño de Arquitectura	15	16	+1
Planificación de Módulos	10	11	+1

La fase de planificación me permitió ajustar mis expectativas en cuanto al tiempo y recursos necesarios, identificando que el análisis y la aclaración de requisitos tomaron más tiempo debido a cambios en las funcionalidades requeridas.

Desarrollo

Diseño

Se diseñó la arquitectura del sistema considerando los requisitos funcionales y no funcionales. El sistema se diseñó utilizando una arquitectura modular y organizada basada en Node.js y Express.js, en la que se implementan varias capas para separar las preocupaciones de lógica de negocio, gestión de datos y presentación. En esta fase, se utilizó un enfoque de Domain-Driven Design (DDD) para dividir la lógica de negocio en módulos específicos, como el módulo de empleados y el módulo de seguridad.

Estructura General del Proyecto:

- **Controllers:** Manejan las solicitudes HTTP e interactúan con los servicios. Por ejemplo, el controlador de empleados gestiona solicitudes de creación, actualización y eliminación de empleados, interactuando con el servicio correspondiente.
- **Services:** Contienen la lógica de negocio, coordinando la obtención de información desde los repositorios y aplicando reglas de negocio. Durante el diseño de los servicios, se definieron métodos para validar datos antes de enviarlos a los repositorios, lo cual redujo los defectos en las fases posteriores.
- **Repositories:** Gestionan la interacción con la base de datos SQLite. Se diseñaron métodos reutilizables para consultar y manipular los datos, lo cual permitió una fácil integración con nuevos módulos que se agregaron más adelante.
- **Models:** Representan las entidades del sistema, como empleados y roles. Durante el diseño de los modelos, se tomaron decisiones sobre la normalización de datos para asegurar la consistencia en la base de datos y evitar la duplicación de información.
- **Helpers y Middlewares:** Funciones de utilidad y componentes para la autenticación y manejo de errores. Por ejemplo, el middleware de autenticación verifica los tokens de los usuarios antes de permitirles acceder a recursos específicos.

- **Swagger:** Utilizado para la documentación de la API, facilitando la comprensión e integración del sistema. Se diseñaron los esquemas de datos en Swagger para asegurar que la documentación estuviera sincronizada con los modelos del sistema.

Diagrama de Componentes y Flujo de Información:

El diseño de la arquitectura también se representó mediante un diagrama de componentes que muestra cómo interactúan los distintos módulos: los Clientes (Frontend) envían solicitudes al API Gateway, que gestiona las solicitudes a través de las capas de Servicios y Datos. Se hizo énfasis en la separación de responsabilidades para facilitar la escalabilidad del sistema y el mantenimiento del código.

Registro de Defectos durante el Diseño:

Actividad	Defecto Encontrado	Descripción del Defecto	Tiempo de Corrección (horas)
Diseño de Roles	Falta de definición clara de permisos	Los permisos de los empleados no estaban bien delimitados	3
Integración Swagger	Problemas con la documentación automática	La configuración inicial de Swagger no mostraba correctamente los endpoints	2
Modelo de Empleados	Estructura inconsistente en el esquema	El esquema inicial de empleados no reflejaba todos los atributos necesarios	1

Codificación

Se realizó la implementación del diseño utilizando Node.js y Express.js. Durante la codificación, se siguieron buenas prácticas como la escritura de pruebas unitarias para cada módulo antes de su integración. Se utilizó Apache JMeter como app de pruebas para verificar la funcionalidad de los servicios y los controladores. Durante la codificación, se registraron defectos relacionados principalmente con la validación de datos y la autenticación de usuarios. El tiempo total invertido en la codificación fue de 27 horas, 2 horas más de lo estimado, debido a la necesidad de corregir errores en la lógica de permisos de usuario.

Actividad	Defecto Encontrado	Descripción del Defecto	Tiempo de Corrección (horas)
Autenticación	Error en la verificación de tokens	Tokens inválidos en rutas específicas	4
Validación de Datos	Falta de validación en campo de estado	El campo de estado no se validaba	3
Gestión de Roles	Problema en la lógica de permisos	Permisos no aplicados correctamente	5

Compilación

La compilación implicó corregir todos los errores de sintaxis e integración antes de proceder con las pruebas. En esta fase se utilizó ESLint para garantizar la calidad del código, asegurando que cumpliera con los estándares establecidos. Se registraron defectos menores durante esta fase, principalmente relacionados con la integración de módulos (por ejemplo, problemas al conectar la autenticación con la gestión de empleados). El tiempo dedicado a la compilación fue de 5 horas.

Actividad	Defecto Encontrado	Descripción del Defecto	Tiempo de Corrección (horas)
Integración de Módulos	Error de conexión entre componentes	Problemas al conectar autenticación y CRUD	1

Pruebas

Se realizaron pruebas exhaustivas para garantizar que cada módulo del sistema funcionara como se esperaba. Las pruebas incluyeron pruebas unitarias, pruebas de integración y pruebas funcionales. Durante esta etapa, se encontraron varios defectos críticos relacionados con el manejo de estados (activo/baja) de los empleados, lo cual me llevó a realizar ajustes significativos en la lógica del sistema. El tiempo total dedicado a las pruebas fue de 22 horas, lo cual excedió la estimación inicial debido a la cantidad de defectos encontrados y su complejidad.

Actividad	Defecto Encontrado	Descripción del Defecto	Tiempo de Corrección (horas)
Manejo de Estados	Error en actualización de estados	El estado del empleado no se actualizaba correctamente	5
Permisos y Roles	Aplicación incorrecta de permisos	Los permisos no eran consistentes en todos los roles	4

3. Postmortem

En la fase de postmortem se llevó a cabo un análisis exhaustivo de los resultados, buscando identificar en qué procesos se insertaron la mayoría de los errores y cuáles fueron las lecciones aprendidas para futuros proyectos. Además, se revisó toda la documentación de defectos y tiempos para corregir cualquier inconsistencia.

Fase	Número de Defectos	Tiempo Total Invertido (horas)	Comentarios sobre el Proceso
Diseño	3	16	La mayoría de los errores se debieron a la falta de cobertura en los casos de prueba iniciales.
Codificación	3	27	Los errores más costosos fueron los relacionados con la validación de datos y la autenticación.

Compilación	1	5	Los defectos fueron menores y relacionados con la integración de módulos.
Pruebas	2	22	Los defectos relacionados con los permisos y estados fueron críticos y tomaron tiempo significativo para corregir.
Total		**70**	

Conclusión

Aplicar los procedimientos del PSP me permitió tener una mejor comprensión de cómo administro mi tiempo y la calidad del código que desarrollo. A lo largo del proyecto de gestión del personal del hotel, los siguientes beneficios y aprendizajes se destacaron:

1. **Mayor Visibilidad sobre el Tiempo Invertido**: La documentación detallada del tiempo invertido en cada fase me permitió comparar las estimaciones iniciales con el tiempo real, lo cual ayudó a ajustar las expectativas para futuras tareas. Identifiqué áreas que requerían más recursos de los previstos, como la fase de pruebas, que excedió el tiempo estimado debido a la cantidad de defectos encontrados y su complejidad.
2. **Mejora de la Calidad del Código**: La implementación de pruebas unitarias desde la fase de codificación y el uso de herramientas de linting como ESLint contribuyeron significativamente a la reducción de errores en el producto final. Los errores más frecuentes estaban relacionados con la validación de datos y la lógica de permisos, lo cual fue corregido gracias al enfoque sistemático de pruebas.
3. **Reducción de Defectos a Través del Análisis**: El registro detallado de defectos me permitió analizar los errores recurrentes y realizar cambios en el proceso para evitarlos. Por ejemplo, al identificar problemas relacionados con los permisos de usuarios y los estados de empleados, implementé cambios en el diseño del sistema para mejorar la validación y reducir estos problemas en fases posteriores.
4. **Escalabilidad y Mantenimiento del Sistema**: El uso de una arquitectura modular basada en **Node.js** y **Express.js** ayudó a crear un sistema más fácil de escalar y mantener. La separación de responsabilidades en distintas capas (controllers, services, repositories) fue clave para garantizar que los cambios en una parte del sistema no afectaran a otras, facilitando la corrección de errores y la incorporación de nuevas funcionalidades.

En resumen, el PSP me ayudó a mejorar mi flujo de trabajo, hacer un uso más efectivo del tiempo y aumentar la calidad del código producido. La documentación detallada de tiempos y defectos me permitió identificar las áreas que requerían mejoras, y las técnicas aprendidas serán muy útiles para proyectos futuros, donde aplicaré de manera más precisa las técnicas de estimación y prevención de errores que desarrollé en este proyecto.

Referencias

Humphrey, W. S. (2005). PSP: A Self-Improvement Process for Software Engineers. Addison-Wesley Professional.

Pressman, R. S., & Maxim, B. R. (2014). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Education.

Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education.://blog.hubspot.es/sales/que-es-pmbok