Análisis de Deuda técnica a través de kata "Supermarket Receipt-Refactoring"

Camilo Alejandro Rojas Salazar
Estudiante Maestría en Informática
Escuela Colombiana de Ingeniería Julio Garavito, Bogotá Colombia
Camilo.rojas-s@mail.escuelaing.edu.co

Resumen

Este artículo pretendió recopilar el análisis de deuda técnica realizado a un código abierto. Se espera que este permita mostrar un ejemplo de los diferentes análisis y conclusiones que pueden evidenciarse en un código sencillo y de esta manera, aplicar el mismo análisis a soluciones o proyectos de mayor tamaño. Se analizaron de manera detallada deudas desde diferentes perspectivas. Los resultados evidencian no solo muestran algunas de las herramientas que se pueden utilizar para identificar diferentes tipos de deuda técnica en el código, sino incluye un análisis de los resultados de los diferentes tipos identificados y posibles formas o estrategias para dares solución.

Palabras clave: Deuda técnica, código abierto, gestión de deuda, mantenimiento de software.

Introducción

El termino deuda técnica ha tenido varios significados desde su aparición, por ejemplo, en su primera mención, se indicó como aquellas tareas que se eligen no realizar ahora pero que pueden generar impactos futuros sino se realizan, es decir, realizar avances que permiten alcanzar objetivos de corto plazo pero que probablemente tienen un impacto negativo a largo plazo (Ciancarini et al., 2019).

Actualmente, en el ciclo de vida de desarrollo de software se ha utilizado para describir diferentes novedades que impactan negativamente en las tareas de implementación, evolución de sistemas y/o obstáculos que limiten el avance de las actividades involucradas en el crecimiento de software. (Rios et al., 2014).

Como forma de solución o de minimización del impacto de la deuda técnica en la calidad del software y en el avance o mantenimiento de una solución de gran complejidad se han establecidos modelos de gestión de deuda técnica que a través de las fases como planear, hacer, verificar y actuar del ciclo de calidad propuesto por Deming (*El Ciclo Deming: En Qué Consiste y Cómo Ayuda En La Gestión de Procesos | Envira Ingenieros Asesores*, 2020), permiten identificar los tipos de deuda técnica más comunes y gestionar estratégicamente la deuda técnica, como lo plantea (Ciancarini & Russo, 2020) en su modelo de gestión de deuda técnica desde una propuesta empírica.

El presente documento realiza un análisis desde la definición actual de deuda técnica, revisando algunos de sus diferentes tipos de deuda al analizar un código abierto, para estructurarlo en 3 partes, incluyendo esta introducción. En la segunda parte se aborda el análisis de las deudas técnicas identificadas en el código abierto incluyendo una o varias estrategias y recomendaciones para disminuir la deuda encontrada y en la tercera parte aborda una serie de conclusiones del análisis realizado y sugerencias para realizar análisis futuros más complejos.

Análisis de Deuda técnica a través de kata "SupermarketReceipt-Refactoring"

A través del análisis que se presenta a continuación, hecho desde diferentes frentes o aspectos se identifican los tipos de deuda técnica de un código abierto, encontrado a través de internet (Bache & Thomas Dave, n.d.); este código fue seleccionado después de varias revisiones y definiciones para realizar el presente trabajo.

Los aspectos que se tienen cuenta para el análisis van desde el Refactoring, identificando Code Smells de la solución, revisando principios de Clean Code que se hayan omitido o no hayan sido tenidos en cuenta en el desarrollo del código. También se analiza Deudas de pruebas o Testing Debt identificando posibles mejoras a nivel de pruebas unitarias, y para finalizar, con el fin de implementar un Modelo de Calidad, se realiza un ejercicio de integración continua a través de Azure DevOps, con la inclusión o uso de herramientas gratuitas de análisis de código como BetterCode (*Better Code Hub*, n.d.) y SonarCloud (SonarSource S.A, n.d.) que permiten realizar comparaciones de resultados de ambos análisis, inclusive teniendo la solución en repositorios diferentes como GitHub y el ya mencionado Azure DevOps, todo esto para complementar la identificación y el respectivo análisis de la deuda en Arquitectura.

2.

Generalidades del código abierto seleccionado

El código abierto presenta la solución en diferentes lenguajes de programación de un supermercado que dispone de un catálogo con diferentes tipos de productos (arroz, manzanas, leche, cepillos de dientes, etc.). Cada producto tiene precio, y el precio total del carrito de compras es el total de todos los precios de los artículos. La solución da un recibo que detalla los artículos que compró, el precio total y los descuentos que se aplicaron. (Rojas Salazar et al., 2022).

Teniendo en cuenta que está en diferentes lenguajes de programación se selecciona el autor selecciona Python teniendo en cuenta su conocimiento sobre este ya que este es mayor que en otros lenguajes de programación.

1. Code Smells detectados

Teniendo en cuenta los indicadores por tipo de deuda técnica en el artículo “Hacia una ontología de los términos de la deuda técnica” (Rios et al., 2014) se identifican los siguientes Code Smells.

1. "Bloaters" en el código "shopping_cart.py" utiliza 'if' para detallar todas las ofertas posibles, dificultando agregar nuevas ofertas como lo muestra la Figura 1. Extracto código “shopping_cart.py

```
            if p in offers.keys():
            offer = offers[p]
            unit_price = catalog.unit_price(p)
            quantity_as_int = int(quantity)
            discount = None
            x = 1
            if offer.offer_type == SpecialOfferType.THREE_FOR_TWO:
                x = 3

            elif offer.offer_type == SpecialOfferType.TWO_FOR_AMOUNT:
                x = 2
                if quantity_as_int >= 2:
                    total = offer.argument * (quantity_as_int / x) + quantity_as_int % 2 * unit_price
                    discount_n = unit_price * quantity - total
                    discount = Discount(p, "2 for " + str(offer.argument), -discount_n)

            if offer.offer_type == SpecialOfferType.FIVE_FOR_AMOUNT:
                x = 5

            number_of_x = math.floor(quantity_as_int / x)
            if offer.offer_type == SpecialOfferType.THREE_FOR_TWO and quantity_as_int > 2:
                discount_amount = quantity * unit_price - (
                            (number_of_x * 2 * unit_price) + quantity_as_int % 3 * unit_price)
                discount = Discount(p, "3 for 2", -discount_amount)
```

Figura 1. Extracto código "shopping_cart.py"

3. "Long Method" en los códigos "receipt_printer.py", y "shopping_cart.py" lo cual dificulta el entendimiento del código al momento de leerlo y de hacer posibles actualizaciones o ingreso de nuevas funcionalidades.
4. "Feature envy" en los códigos "receipt.py", "receipt_printer.py" y "shopping_cart.py" no utiliza los métodos propios, sino que utiliza más los métodos del código "model_objects.py", "receipt_printer.py" y "model_objects.py" respectivamente. Ejemplo de esto lo vemos en Figura 2. Extracto código "receipt_printer.py"

```
    def print_receipt(self, receipt):
        result = ""
        for item in receipt.items:
            receipt_item = self.print_receipt_item(item)
            result += receipt_item

        for discount in receipt.discounts:
            discount_presentation = self.print_discount(discount)
            result += discount_presentation

        result += "\n"
        result += self.present_total(receipt)
        return str(result)
```

Figura 2. Extracto código "receipt_printer.py"

Técnicas por utilizar para solucionar Code Smells

Teniendo en cuenta lo sugerido por (*Refactoring: Clean Your Code*, n.d.) se sugiere lo siguiente.

1. Refactoring al archivo "receipt_printer.py" realizando la extracción de método, moviendo la lógica de salto de línea a una nueva clase.
2. Refactoring al archivo "shopping_cart.py" realizando la extracción de método, moviendo los cálculos de ofertas a una nueva clase llamada detalle.
3. Refactoring al detalle de "shopping_cart.py" modificando la lógica

del método para evitar la dependencia de los "ifs".

2. *Características del Clean code no cumplidas*

- Código enfocado. El código de "shopping_cart.py" contiene la cantidad de productos y adicionalmente realiza los cálculos de descuentos. Por ejemplo

```
@property
    def product_quantities(self):
        return self._product_quantities

    def add_item_quantity(self, product, quantity):
        self._items.append(ProductQuantity(product, quantity))
        if product in self._product_quantities.keys():
            self._product_quantities[product] = self._product_quantities[product] + quantity
        else:
            self._product_quantities[product] = quantity

    def handle_offers(self, receipt, offers, catalog):
        for p in self._product_quantities.keys():
            quantity = self._product_quantities[p]
            if p in offers.keys():
                offer = offers[p]
                unit_price = catalog.unit_price(p)
                quantity_as_int = int(quantity)
                discount = None
                x = 1
                if offer.offer_type == SpecialOfferType.THREE_FOR_TWO:
                    x = 3

                elif offer.offer_type == SpecialOfferType.TWO_FOR_AMOUNT:
                    x = 2
                    if quantity_as_int >= 2:
                        total = offer.argument * (quantity_as_int / x) + quantity_as_int % 2 * unit_price
                        discount_n = unit_price * quantity - total
                        discount = Discount(p, "2 for " + str(offer.argument), -discount_n)

                if offer.offer_type == SpecialOfferType.FIVE_FOR_AMOUNT:
                    x = 5
```

Figura 3. Extracto código "shopping_cart.py"

- Entendible. El código de "shopping_cart.py" no tiene un código simple incumpliendo el principio KISS (Keep it Simple, ~~Stupid~~). Como se evidencia en Figura 3. Extracto código "shopping_cart.py"

- Escalable. El código de "shopping_cart.py" no tiene un código escalable en la lógica de descuentos como se muestra en la Figura 4 Extracto clase ofertas en la sección "shopping_cart.py"

```
if offer.offer_type == SpecialOfferType.THREE_FOR_TWO and quantity_as_int > 2:
    discount_amount = quantity * unit_price - (
                (number_of_x * 2 * unit_price) + quantity_as_int % 3 * unit_price)
    discount = Discount(p, "3 for 2", -discount_amount)

if offer.offer_type == SpecialOfferType.TEN_PERCENT_DISCOUNT:
    discount = Discount(p, str(offer.argument) + "% off",
                        -quantity * unit_price * offer.argument / 100.0)

if offer.offer_type == SpecialOfferType.FIVE_FOR_AMOUNT and quantity_as_int >= 5:
    discount_total = unit_price * quantity - (
                offer.argument * number_of_x + quantity_as_int % 5 * unit_price)
    discount = Discount(p, str(x) + " for " + str(offer.argument), -discount_total)
```

Figura 4 Extracto clase ofertas en la sección "shopping_cart.py"

- Abstracción. El código de "shopping_cart.py" utiliza un método largo para el cálculo de los descuentos. Adicionalmente, El código de "receipt_printer.py" utiliza una clase extensa para imprimir el recibo que contiene la lista de productos y la estructura que el recibo debe tener.

3. *Principios de Programación no cumplidos en el código*

KISS

El principio traduce Mantenlo Simple (Keep it Simple, ~~Stupid~~). No todo el código es simple, en especial el código de "shopping_cart.py" ya que incluye la lógica de los productos y sus cantidades y adicionalmente, la lógica de descuentos con muchos “if” de por medio.

SOLID

Solid es un acróstico referente a mantener características de segregación o dependencia de objetos que no utilizan y responsabilidad unitaria de cada parte del código, extensión de código cerrando las modificaciones de métodos o clases. Teniendo en cuenta lo anterior, de la solución analizada se puede deducir lo siguiente:

- A como está con el código hoy, en especial el código de "shopping_cart.py" está abierto a modificaciones debido a la forma en que se maneja la lógica de descuentos. Esto también implica que la clase de este código esté afectada ya que hace varias cosas, tiene la lógica de cantidad y productos y la lógica de descuento.
- En la clase del código "receipt_printer.py" incluye el formato del recibo y recibe la información de productos y cantidades.

4. *Diagnóstico Inicial Testing Debt*

La solución cuenta con una sola prueba unitaria "test ten percent discount" la cual tiene como objetivo probar uno de los escenarios de descuento o de oferta ofrecidos por el supermercado como lo muestra Figura 5. Extracto código de pruebas unitarias. Esta prueba tiene las siguientes características y falencias:

```
def test_ten_percent_discount():
  catalog = FakeCatalog()
  toothbrush = Product("toothbrush", ProductUnit.EACH)
  catalog.add_product(toothbrush, 0.99)

  apples = Product("apples", ProductUnit.KILO)
  catalog.add_product(apples, 1.99)

  teller = Teller(catalog)
  teller.add_special_offer(SpecialOfferType.TEN_PERCENT_DISCOUNT, toothbrush, 10.0)

  cart = ShoppingCart()
  cart.add_item_quantity(apples, 2.5)

  receipt = teller.checks_out_articles_from(cart)

  assert 4.975 == pytest.approx(receipt.total_price(), 0.01)
  assert [] == receipt.discounts
  assert 1 == len(receipt.items)
  receipt_item = receipt.items[0]
  assert apples == receipt_item.product
  assert 1.99 == receipt_item.price
  assert 2.5 * 1.99 == pytest.approx(receipt_item.total_price, 0.01)
  assert 2.5 == receipt_item.quantity
```

Figura 5. Extracto código de pruebas unitarias.

Prácticas Cumplidas

- Utiliza un estándar de nombramiento (Para el uso de pytest la prueba unitaria debe comenzar con la palabra "test").

Prácticas No Cumplidas

- No utiliza prácticas de Clean Code en sus pruebas Unitarias (UT).
- Solo valida un escenario de negocio, descuento del 10%.
- Tiene muchos asserts en su única prueba (7).
- La prueba unitaria depende del archivo "fake_catalog.py" que en teoría es independiente de la solución.
- No utiliza el patrón AAA (Arrange, Act, Assert).
- Dado que solo existe una prueba unitaria, no se promueve un alto nivel de cobertura (Coverage).

Propuesta de Pruebas Unitarias

Teniendo en cuenta el diagnóstico inicial, se propone 3 pasos para mejorar el Coverage de las pruebas unitarias.

Paso 1. Estandarizar la prueba actual con el Patrón AAA (Arrange, Act, Assert) y hacer la respectiva división de Asserts o dejar los que dan mayor alcance y utilizar las prácticas de Clean Code. Ver Figura 6 Prueba Unitaria ajustada de acuerdo al Patrón AAA

```
def test_ten_percent_discount():
  #ARRANGE
  catalog = FakeCatalog()
  toothbrush = Product("toothbrush", ProductUnit.EACH)
  apples = Product("apples", ProductUnit.KILO)
  catalog.add_product(toothbrush, 0.99)
  catalog.add_product(apples, 1.99)
  teller = Teller(catalog)

  #ACT
  teller.add_special_offer(SpecialOfferType.TEN_PERCENT_DISCOUNT, toothbrush, 10.0)
  cart = ShoppingCart()
  cart.add_item_quantity(apples, 3)
  receipt = teller.checks_out_articles_from(cart)
  receipt_item = receipt.items[0]
  ResponseExpected = 3 * 1.99

  #ASSERT
  assert [] == receipt.discounts
  assert 1 == len(receipt.items)
  assert apples == receipt_item.product
  assert ResponseExpected == pytest.approx(receipt.total_price(), 0.01)
  assert ResponseExpected == pytest.approx(receipt_item.total_price, 0.01)
```

Figura 6 Prueba Unitaria ajustada de acuerdo al Patrón AAA

Paso 2. Continuando la estrategia de pruebas propuesta en la solución (probar los escenarios de descuentos) se necesita complementar las pruebas por escenario de Oferta especial. Por ejemplo:

- Test_SingleProduct_NoDiscount()

```
def test_SingleProduct_NoDiscount():
  qty = 5
  price = 2.99
  #ARRANGE
  catalog = FakeCatalog()
  oranges = Product("oranges", ProductUnit.KILO)
  catalog.add_product(oranges, price)
  teller = Teller(catalog)


  #ACT
  teller.add_special_offer(SpecialOfferType.FIVE_FOR_AMOUNT, oranges, 5.0)

  cart = ShoppingCart()
  cart.add_item_quantity(oranges,qty)

  receipt = teller.checks_out_articles_from(cart)
  receipt_item = receipt.items[0]
  ResponseExpected = qty*price

  #ASSERT
  assert ResponseExpected == pytest.approx(receipt_item.total_price, 0.01)
  assert oranges == receipt_item.product
```

Figura 7 Propuesta Test_SingleProduct_NoDiscount()

Como el caso visto en Figura 7 Propuesta Test_SingleProduct_NoDiscount(), se proponen incluir los siguientes casos adicionales

- test_two_for_amount_discount()
- test_five_for_amount_discount()
- test_three_for_two_discount()

Paso 3. Por último, se considera necesario agregar pruebas unitarias por

sección. La solución propuesta tiene la ventaja de estar divida en varias partes o secciones (teller, Shopping_cart, model_objects, etc.) lo recomendable es que por cada parte incluir al menos una prueba unitaria para cada método que incluya lógica de negocio. Ver Figura 8 Propuesta para test Test_teller_qty()

```
def test_teller_qty():
 #ARRANGE
 catalog = FakeCatalog()
 apples = Product("apples", ProductUnit.KILO)
 catalog.add_product(apples, 1.99)
 cart = ShoppingCart()

 #ACT
 teller = Teller(catalog)
 cart.add_item_quantity(apples, 3)
 receipt = teller.checks_out_articles_from(cart)
 Receipt_item = receipt.items[0]
 ReceiptExpected = 3

 #ASSERT
 assert ReceiptExpected == Receipt_item.quantity
```

Figura 8 Propuesta para test Test_teller_qty()

Mejoras para reducción de deuda

Con el fin de seguir con el proceso de reducción de deuda técnica en el código se dan las siguientes recomendaciones.

- Ser más organizado con el código (incluyendo las UT).
- Utilizar las prácticas de clean code hasta en las UT.
- Segmentar mejor las pruebas

5. *Modelo de calidad*

Pasos preliminares

Para realizar el análisis a través de (*Better Code Hub*, n.d.) se debe tener cuenta en la plataforma o se puede utilizar la cuenta de GitHub. Se puede utilizar en versión gratuita la cual permite realizar análisis ilimitados pero las soluciones que estén como públicas o la cuenta paga la cual permite el análisis a soluciones privadas. Cabe mencionar que los planes pagos tienen una versión de prueba por un tiempo limitado.

Una vez seleccionado el plan, la herramienta precarga la información de todos los repositorios de tu cuenta GitHub. Selecciona el repositorio correspondiente y ejecuta el análisis.

Resultado del Análisis

Después de la ejecución del análisis en la solución a través de la plataforma BetterCodeHub (Ver Figura 9 Resultados Better Code Hub), el cual presenta un análisis en múltiples aspectos (10 aspectos), se detecta que el código cumple con los siguientes ítems positivos.

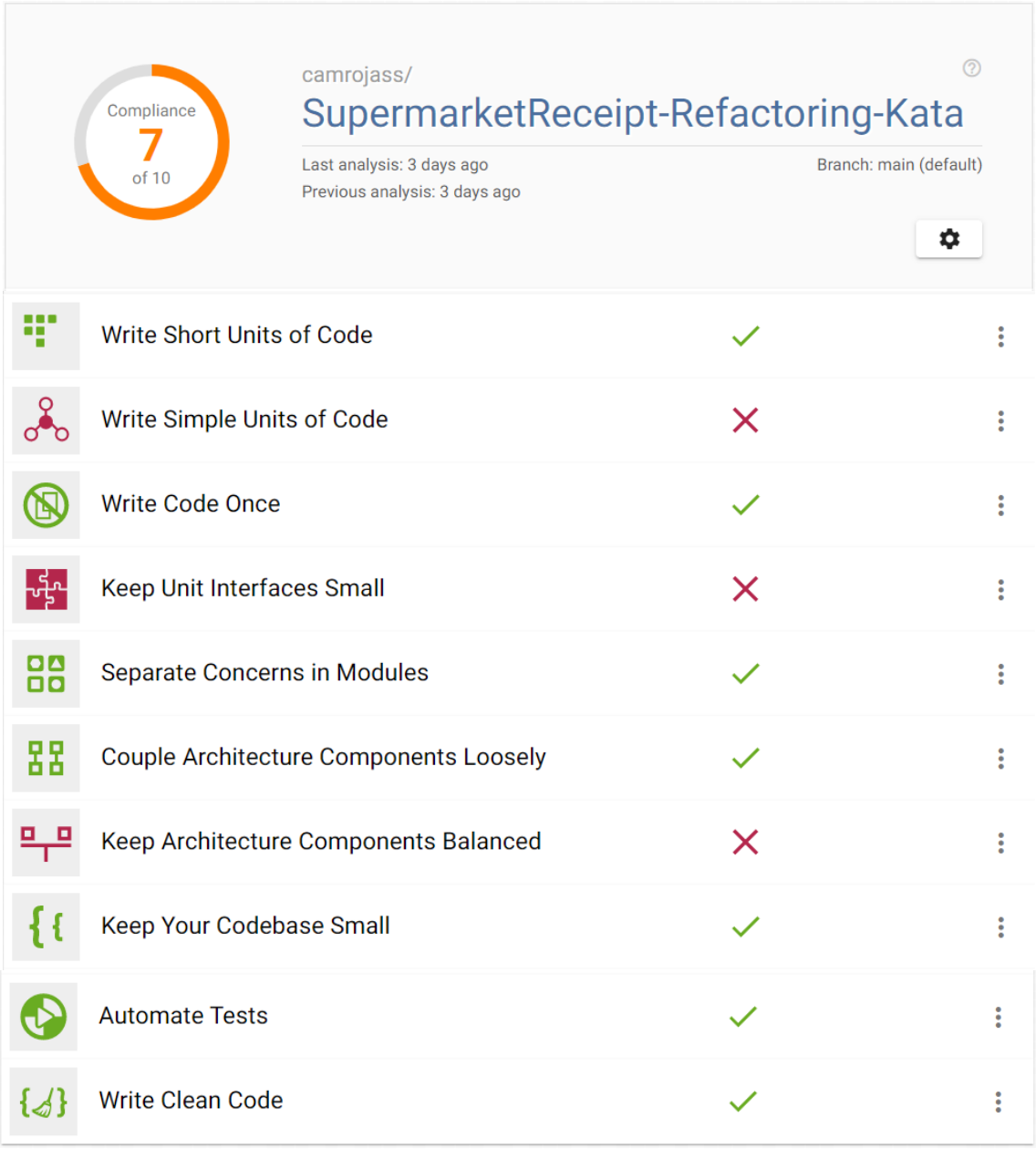

Figura 9 Resultados Better Code Hub

- *Escribir el código en pequeñas unidades*. Esto indica que el código en general está seccionado en pequeñas unidades.
- *Escribir código una sola vez (No repetido)*. El código no tiene secciones repetidas.
- *Separar código por módulos*. El código está separado por módulos (Modelo de Objetos, Carro de compras, Recibido, etc.)
- *Componentes de Arquitectura libres*. Se debe tener en cuenta que su estructura la solución es un solo código lo cual lo hace independiente y sin necesidad de otros componentes. Se debe tener en cuenta para el momento de agregar

nuevo código que se mantenga esa condición.

- *Mantener el código pequeño.* El código está divido en secciones pequeñas.
- *Pruebas automatizadas.* El código cuenta con pruebas automatizadas.
- *Código limpio.* De acuerdo con la plataforma no se encuentran secciones del proyecto con código sucio.

Sin embargo, tiene por mejorar los siguientes ítems

- *Escribir unidades pequeñas.* Esto debido a que una parte de la solución es muy extensa, específicamente en el módulo de "*Shopping_Car.py*" al incluir las ofertas de descuento. Ver Figura 10 Extracto código descuentos "shopping_car.py"

```
def handle_offers(self, receipt, offers, catalog):
    for p in self._product_quantities.keys():
        quantity = self._product_quantities[p]
        if p in offers.keys():
            offer = offers[p]
            unit_price = catalog.unit_price(p)
            quantity_as_int = int(quantity)
            discount = None
            x = 1
            if offer.offer_type == SpecialOfferType.THREE_FOR_TWO:
                x = 3

            elif offer.offer_type == SpecialOfferType.TWO_FOR_AMOUNT:
                x = 2
                if quantity_as_int >= 2:
                    total = offer.argument * (quantity_as_int / x) + quantity_as_int % 2 * unit_price
                    discount_n = unit_price * quantity - total
                    discount = Discount(p, "2 for " + str(offer.argument), -discount_n)

            if offer.offer_type == SpecialOfferType.FIVE_FOR_AMOUNT:
                x = 5

            number_of_x = math.floor(quantity_as_int / x)
            if offer.offer_type == SpecialOfferType.THREE_FOR_TWO and quantity_as_int > 2:
                discount_amount = quantity * unit_price - (
                            (number_of_x * 2 * unit_price) + quantity_as_int % 3 * unit_price)
                discount = Discount(p, "3 for 2", -discount_amount)
```

Figura 10 Extracto código descuentos "shopping_car.py"

- *Contiene unidades de interfase pequeñas.* Se debe a que en pequeñas secciones de código se solicita muchos parámetros como, por ejemplo, se muestra en la Figura 11. Extracto "Model_object.py"

```
def __init__(self, product, quantity, price, total_price):
    self.product = product
    self.quantity = quantity
    self.price = price
    self.total_price = total_price
```

Figura 11. Extracto "Model_object.py"

- *Balance de los componentes de arquitectura*. De acuerdo con el análisis la solución, al ser un único componente de 176 líneas no es balanceado. Cabe mencionar que el código no complejo por lo que no es estrictamente un punto delicado. Se debe tener en cuenta al aplicar nuevas funciones que los componentes no superen las 170 líneas por componente.

6. *Integración Continua*

Pasos preliminares

Para la integración a través de Azure DevOps se recomienda seguir el paso a paso sugerido en el proyecto en GitHub por (Rojas, 2022b)

Este paso a paso se hizo basado en las recomendaciones de los artículos dispuestos a través del foro Azure DevOps Labs (*Integrate Your GitHub Projects With Azure Pipelines | Azure DevOps Hands-on-Labs*, n.d.) y las definiciones dadas por Microsoft en su documentación (*Compilación y Prueba de Aplicaciones de Python - Azure Pipelines | Microsoft Docs*, n.d.)

Resultados de la integración

A continuación, como lo muestran las figuras Figura 12. Pipelines de Azure DevOps configurados. y Figura 13. Detalle del Pipeline relacionado a GitHub, se evidencia el proceso exitoso de la implementación del proyecto en Azure.

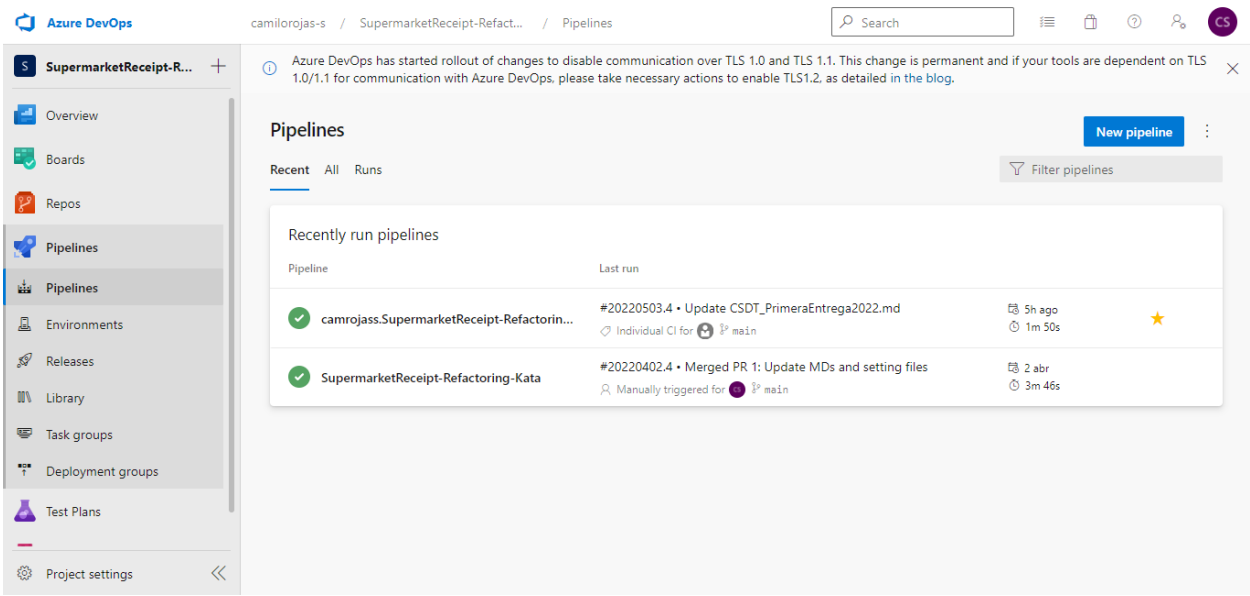

Figura 12. Pipelines de Azure DevOps configurados.

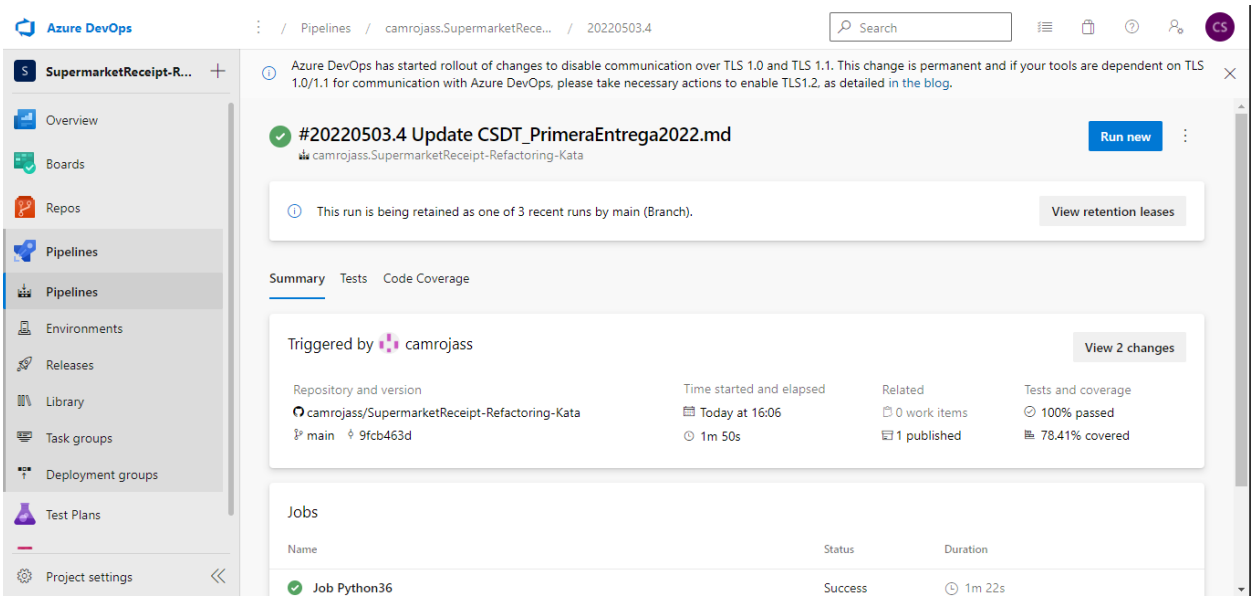

Figura 13. Detalle del Pipeline relacionado a GitHub

Como lo muestra la Figura 14. Resultados de Cobertura identificados a través de los agentes de Azure DevOps la integración a través permite obtener resultados de las pruebas unitarias realizadas y adicionalmente un indicador de cobertura del código respecto a las pruebas unitarias que tenga la solución.

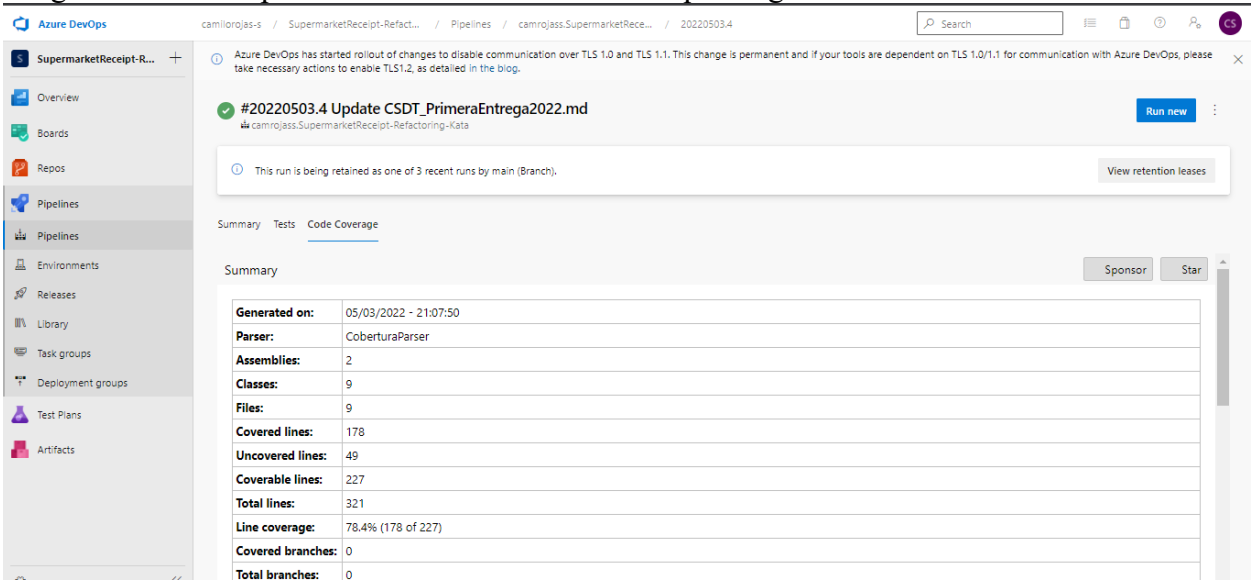

Figura 14. Resultados de Cobertura identificados a través de los agentes de Azure DevOps

Integración con SonarCloud

Para la integración con SonarCloud se utilizaron dos métodos de integración, el primero a través de Azure el cual implica que el proyecto sea público desde la plataforma para la visualización de resultados y el segundo a través de GitHub que es más sencillo, pero tiene sus limitaciones si el proyecto no ha tenido modificaciones recientes en los últimos 30 días. Todos los pasos para ambos métodos de integración se encuentran disponibles en (Rojas, 2022a)

Resultados SonarCloud

Al hacer la integración a través de Azure DevOps se obtuvieron los siguientes resultados como lo evidencia la Figura 15. Resumen resultado SonarCloud

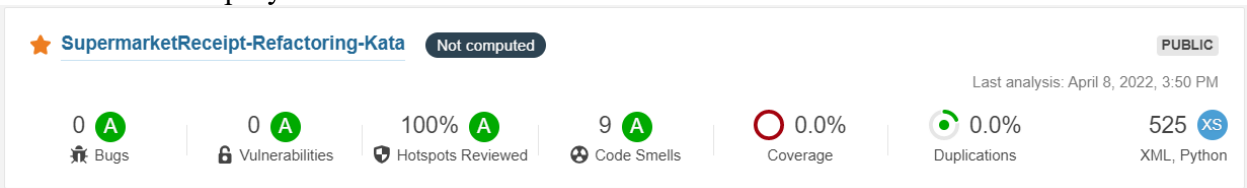

Figura 15. Resumen resultado SonarCloud

Como lo muestra la imagen SonarCloud evalúa la carpeta de las pruebas unitarias como parte conjunta del código esto debido a que no se encuentra en la raíz del proyecto.

- *Descripción del proyecto.* En la gráfica de Coverage vs Deuda técnica evidencia que en general el Code Smells detectado no es de alto impacto (color verde) lo cual evidencia que en general el proyecto tiene cosas por mejorar que no son de alto impacto. Por otro lado, y teniendo en cuenta que está contemplando la carpeta de pruebas como una carpeta de código y no de pruebas unitarias, indica que el

principal trabajo a realizar en esta carpeta. Ver Figura 16. Gráfica Coverage vs Code Smells

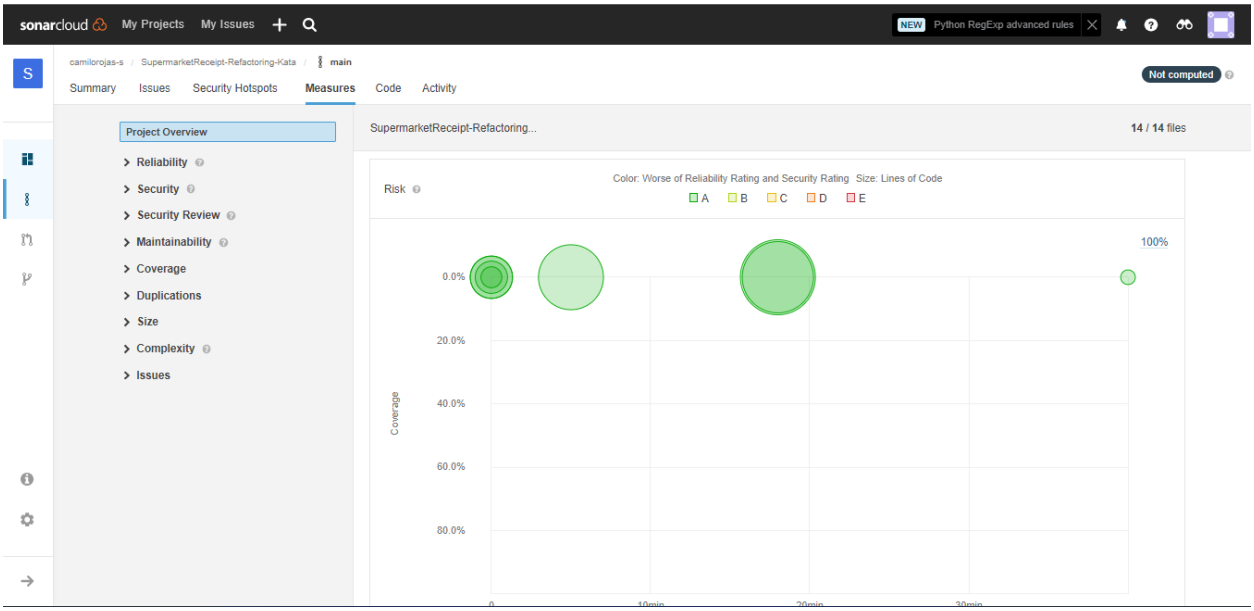

Figura 16. Gráfica Coverage vs Code Smells

- *Code Smells detectado.* Los Code Smells detectados por Sonar se debe principalmente al uso de caracteres especiales o el guardado de información en variables poco claras (i, x, etc.) en especial en las pruebas unitarias. Sonar hace la aclaración que la prioridad son las variables debido a que son errores críticos, sin embargo, se evidencia que la calificación general del proyecto respecto al ítem es calificación A. Figura 17. Code Smells detectados en Sonar Cloud

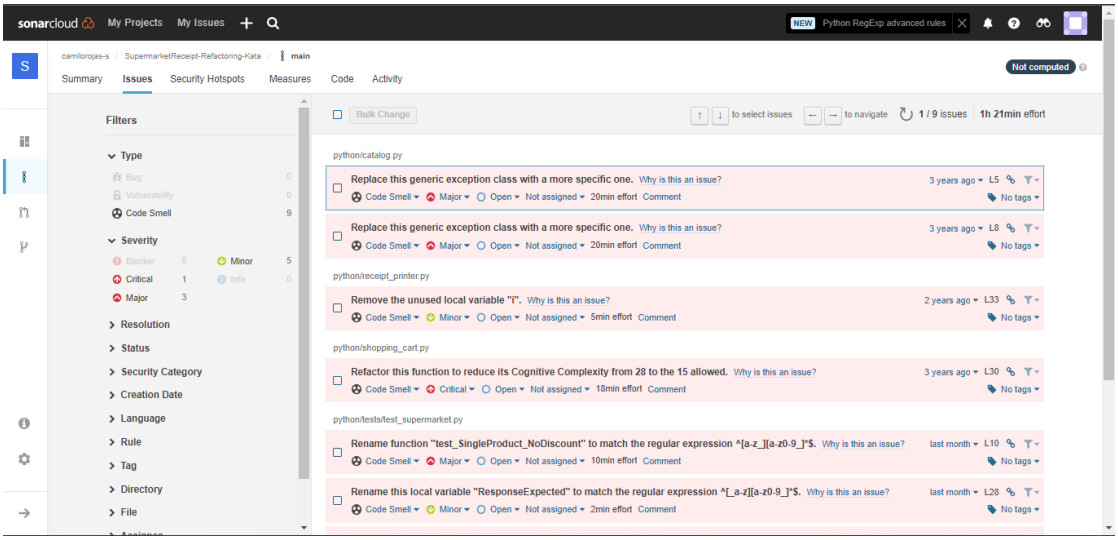

Figura 17. Code Smells detectados en Sonar Cloud

- *Indicador de Deuda.* El índice de deuda indica que el porcentaje de deuda técnica con respecto a la solución completa es pequeño y optimo debido que solo es un 1.1% del código que tiene deuda respecto al total de las líneas de código. Figura 18. índice de deuda identificado por Sonar Cloud

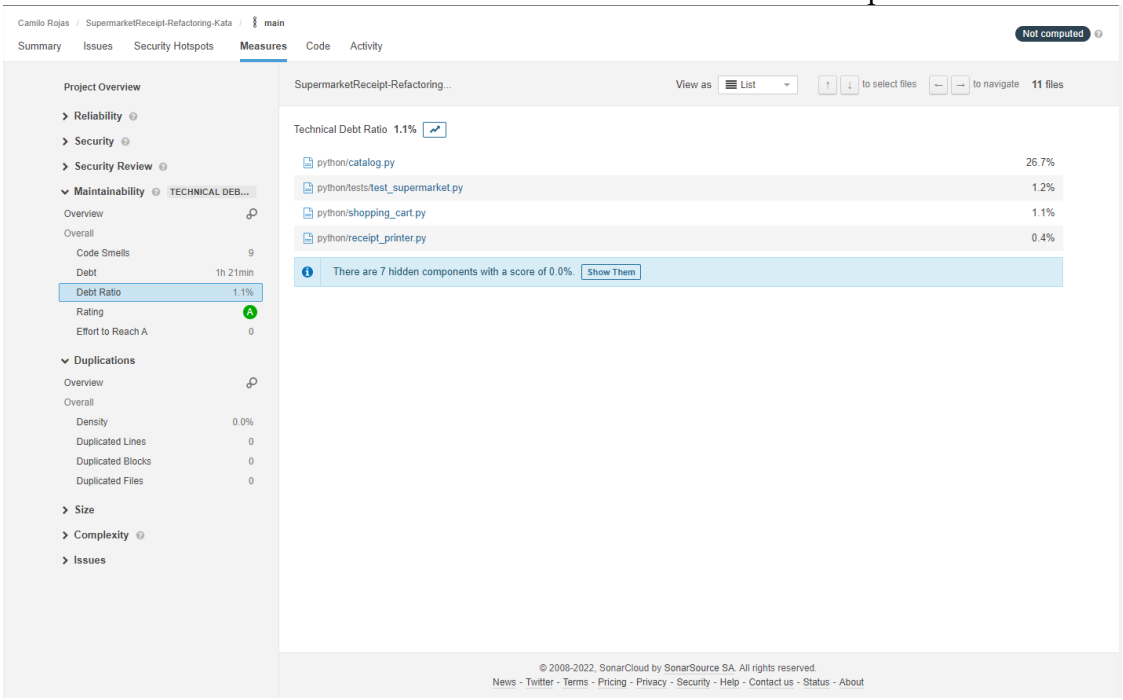

Figura 18. índice de deuda identificado por Sonar Cloud

Cabe mencionar que, dado que se hizo la integración continua con Azure DevOps, y existen dos ramas allí, se ve el doble de líneas de código. En general la solución tiene buena calificación respecto a su tamaño, lo cual la cataloga como solución buena debido al rating A en los diferentes aspectos analizados por Sonar.

7. *Atributos de calidad y Deuda en Arquitectura*

Teniendo en cuenta los resultados obtenidos en los diferentes análisis realizados a través de SonarCloud, se hace necesario considerar los siguientes escenarios de negocio como parte de la mejora continua de la arquitectura del proyecto de acuerdo con el modelo ATAM (Kazman et al., 2000).

Escenario No. 1

Se evidencia en los diferentes análisis que una de las actuales limitaciones del sistema es la cobertura de código diversa en los diferentes elementos o componentes del proyecto, por tal razón se hace necesario establecer un estándar mínimo para todo el código. Ver Tabla 1 Escenario No. 1 Atributos de calidad de Arquitectura

Tabla 1 Escenario No. 1 Atributos de calidad de Arquitectura

Refinamiento de Escenario No. 1		
Escenario		El administrador del supermercado desea agregar un nuevo descuento con motivo de madrugón en productos seleccionados un día al año de 6 a 9 de la mañana.
Objetivo de Negocio		Agregar al nicho nuevos clientes y realizar ventas superiores a las diarias
Atributo de calidad relevante		Mantenibilidad
Componentes del escenario	**Estímulo**	Nuevo descuento
	Origen del Estímulo	Estrategia para obtener mayores ganancias
	Ambiente	Desarrollo y validación del escenario de descuento
	Artefacto	Shopping_cart.py
	Respuesta	El sistema debe realizar el descuento solo en el horario seleccionado y sobre los productos seleccionados para la promoción sin errores.
	Medida de respuesta	2 horas.
Pregunta		¿Cuánto tiempo toma actualizar el sistema para que permita cumplir con el escenario de negocio planteado sin errores?
Novedad		Puede necesitar mejorar el código relacionado a las ofertas existentes para disminuir la deuda técnica del proyecto.

Fuente: Elaboración propia

Escenario No. 2

Teniendo en cuenta que las diferentes ofertas se encuentran en un código espagueti, se hace evidente que se establezca un escenario de nuevos descuentos a aplicar con consideraciones más específicas como lo es el descuento de Madrugón o Trasnochón que los supermercados realizan en fechas especiales del año.

Tabla 2 Escenario No. 2 Atributos de calidad en Arquitectura

Tabla 2 Escenario No. 2 Atributos de calidad en Arquitectura

Refinamiento de Escenario No. 2		
Escenario		Inclusión de nuevos cajeros en el sistema en temporadas de alta demanda
Objetivo de Negocio		Mejorar el tiempo de servicio del supermercado debido al aumento de clientes en ciertas horas de alta demanda.
Atributo de calidad relevante		Escalabilidad, rendimiento
Componentes del escenario	**Estímulo**	Mayor cantidad de clientes
	Origen del Estímulo	Mejor publicidad del supermercado y fidelización
	Ambiente	Operación bajo alta demanda
	Artefacto	Teller.py
	Respuesta	El sistema debe permitir
	Medida de respuesta	Disminuir el tiempo de servicio por cliente (2 a 3 minutos por cliente)
Pregunta		¿Cómo puedo aumentar la cantidad de cajeros que atienden en determinadas horas del día sin afectar el rendimiento del sistema?
Novedad		Se necesita ampliar la condición la cantidad de cajeros en horarios de demanda sin afectar el rendimiento del sistema.

Fuente: Elaboración propia

Escenario No. 3

Dentro de las consideraciones a tener en cuenta está el aumento de clientes que puede tener un supermercado en horas pico lo cual hace inevitable que se agregue más componentes para mejorar el tiempo de servicio, sin embargo, esto implica realizar escalamientos del sistema manteniendo el rendimiento actual.

Tabla 3 Escenario No. 3 Atributos de Calidad en Arquitectura

Refinamiento de Escenario No. 3		
Escenario		Se evidencia que se debe mejorar la cobertura de código para alguno de los componentes del proyecto como la impresora del recibo y el recibo.
Objetivo de Negocio		Mejorar la cobertura del código promedio entre los diferentes componentes del proyecto.
Atributo de calidad relevante		Testeabilidad.
Componentes del escenario	**Estímulo**	Mejorar la calidad del proyecto sin aumentar de forma drástica el costo de paso a
	Origen del Estímulo	Análisis de cobertura en SonarCloud y Azure DevOps
	Ambiente	Desarrollo de software
	Artefacto	Pruebas unitarias
	Respuesta	El sistema debe conseguir un promedio de cobertura de código alto en sus diferentes componentes
	Medida de respuesta	La cobertura de código de todos sus componentes debe estar sobre el 80%
Pregunta		¿Cómo puedo aumentar la cobertura de código para mejorar el promedio total de cobertura del sistema?
Novedad		Se necesita incluir más pruebas unitarias para cada uno de los componentes del proyecto

Fuente: Elaboración propia

Conclusiones

Teniendo en cuenta los diferentes aspectos analizados, se evidencia que, de un código o solución, por más sencillo o pequeño que sea se pueden identificar y analizar diferentes escenarios y realizar un completo análisis de deuda técnica.

La solución analizada en general tiene deudas técnicas que se pueden considerar pequeñas pero que no pueden ser omitidas en caso de extender el proyecto a escenarios propuestos como la inclusión de nuevos descuentos que impliquen tener en cuenta aspectos como el horario o la ampliación a dos o n cajas para la recepción de más productos al mismo tiempo.

El principal objetivo para solucionar, de acuerdo con los diferentes análisis evidenciados es la mejora a nivel de pruebas unitarias, esto dado que no cumplían con patrones de diseño de pruebas y los escenarios iniciales no contemplan la cobertura suficiente del código.

Como segundo ítem en prioridad a dar solución es a la distribución y organización de los descuentos, si bien este proyecto solo implicaba código para lo que corresponde a aplicación, en una extensión del proyecto, estos descuentos se sugiere que no estén en el código sino en una base de datos de la aplicación, resolviendo así un gran porcentaje de los code Smells detectados a través de las diferentes herramientas.

Es importante resaltar que los atributos de calidad en arquitectura se deben extender a analizar toda la integridad de la solución, dado que el proyecto analizado solo tenía una estructura relacionada a lo que sería la aplicación, hay que tener en cuenta que se debe evaluar los demás componentes como lo sería las bases de datos, integración con otros componentes, etc.

Referencias

SonarSource S.A. (n.d.). *Automatic Code Review, Testing, Inspection & Auditing | SonarCloud*. Retrieved May 1, 2022, from https://sonarcloud.io/

Better Code Hub. (n.d.). Retrieved May 1, 2022, from https://bettercodehub.com/

Rojas Salazar, C. A. (camrojass), Bache, E. (emilybache), & Thomas, D. (2022, May). *SupermarketReceipt-Refactoring-Kata*. https://github.com/camrojass/SupermarketReceipt-Refactoring-Kata

Ciancarini, P., Nuzzolese, A. G., Presutti, V., & Russo, D. (2019). *SQuAP-Ont: an Ontology of Software Quality Relational Factors from Financial Systems*. https://doi.org/10.3233/SW-200372

Ciancarini, P., & Russo, D. (2020). *The Strategic Technical Debt Management Model: An Empirical Proposal* (pp. 131–140). https://doi.org/10.1007/978-3-030-47240-5_13

Compilación y prueba de aplicaciones de Python - Azure Pipelines | Microsoft Docs. (n.d.). Retrieved May 2, 2022, from https://docs.microsoft.com/es-mx/azure/devops/pipelines/ecosystems/python?view=azure-devops

El ciclo Deming: en qué consiste y cómo ayuda en la gestión de procesos | Envira Ingenieros Asesores. (2020, August 14). https://envira.es/es/el-ciclo-deming-que-consiste-y-como-ayuda-gestion-procesos/

Bache, E., & Thomas Dave. (n.d.). *emilybache/SupermarketReceipt-Refactoring-Kata: This is a refactoring kata based on the one described in http://codekata.com/kata/kata01-supermarket-pricing/*. Retrieved May 1, 2022, from https://github.com/emilybache/SupermarketReceipt-Refactoring-Kata

Integrate Your GitHub Projects With Azure Pipelines | Azure DevOps Hands-on-Labs. (n.d.). Retrieved May 2, 2022, from

https://www.azuredevopslabs.com/labs/azuredevops/github-integration/

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for Architecture Evaluation*.

Refactoring: clean your code. (n.d.). Retrieved May 2, 2022, from https://refactoring.guru/es/refactoring

Rios, N., Ribeiro, L., Caires, V., Mendes, T., & Spínola, R. (2014). Towards an Ontology of Terms on Technical Debt. *Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014*, 1–7. https://doi.org/10.1109/MTD.2014.9

Rojas, C. A. (2022a, May 2). *CSDT_ArchitecturalSmells.md*. https://github.com/camrojass/SupermarketReceipt-Refactoring-Kata/blob/main/python/CSDT_ArchitecturalSmells.md

Rojas, C. A. (2022b, May 2). *CSDT_Continuos Integration.md*. https://github.com/camrojass/SupermarketReceipt-Refactoring-Kata/blob/main/python/CSDT_ContinuousIntegration.md