CARRERA DE COMPUTACIÓN

INFORME

1. Datos informativos

1.1.Módulo: 1

1.2.Nivel: Séptimo

1.3.Fecha: 25 de febrero de 2025

1.4.Nombres: Aupas Antony, Baraja Cristian, Basantes Geovanny

1.5.Tema: Aplicación de RUP al Proyecto de Explicabilidad de Modelos de IA en NLP

2. Objetivo

Describir el proceso de desarrollo e implementación un sistema de explicabilidad de modelos de Inteligencia Artificial (IA) aplicados al Procesamiento de Lenguaje Natural (NLP), utilizando las técnicas LIME y SHAP, mediante la aplicación del Rational Unified Process (RUP), para garantizar la transparencia, comprensión y confianza en las decisiones tomadas por los modelos de IA.

3. Contenido

El desarrollo de software para el sistema de explicabilidad de modelos NLP seguirá las fases de la metodología RUP, que según Kruchten (2004) "proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo" (p. 17).

El Rational Unified Process (RUP) es una metodología ágil que proporciona una guía clara para el desarrollo de software mediante un ciclo iterativo en cuatro fases: Inicio, Elaboración, Construcción y Transición. En este proyecto, se utilizará RUP para gestionar y organizar cada fase del desarrollo del sistema de explicabilidad, asegurando que se cumplan los requisitos técnicos y funcionales de manera efectiva y escalable.

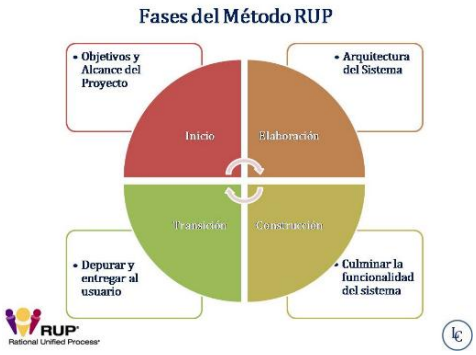

Figura 1. Fases del Método RUP

3.1.Desarrollo iterativo

3.1.1. Fase de Inicio (Inception)

Visión del producto: Sistema de explicabilidad para modelos de IA en procesamiento de lenguaje natural que permita comprender cómo toman decisiones estos modelos (Arlow y Neustadt, 2005).

Casos de uso clave: Análisis de textos, generación de explicaciones visuales, detección y mitigación de sesgos (Arriaga et al., 2020).

Riesgos principales: Complejidad en la implementación de técnicas de explicabilidad, integración con modelos pre-entrenados (Boehm, 1991).

3.1.2. Fase de Elaboración (Elaboration)

Arquitectura base: Implementación de backend en Django, frontend en React, bases de datos híbridas (MongoDB y PostgreSQL) como recomiendan Fowler y Scott (2000).

Refinamiento de requisitos: Especificación detallada de técnicas de explicabilidad (LIME y SHAP) siguiendo el enfoque de Ribeiro et al. (2016).

Plan de proyecto: Organización del desarrollo en iteraciones con metodología SCRUM (Schwaber y Sutherland, 2017).

3.1.3. Fase de Construcción (Construction)

Implementación iterativa: Desarrollo modular comenzando por la integración de modelos NLP, seguido por las técnicas de explicabilidad (Jacobson et al., 1999).

Pruebas continuas: Implementación de pruebas unitarias para cada componente y pruebas de integración (Kroll y Kruchten, 2003).

Gestión de cambios: Uso de Git para control de versiones y GitHub para colaboración (Loeliger y McCullough, 2012).

3.1.4. Fase de Transición (Transition)

Despliegue: Implementación en servidores cloud (AWS o Google Cloud) usando contenedores Docker (Humble y Farley, 2010).

Evaluación del usuario: Pruebas de aceptación con usuarios finales para verificar la utilidad de las explicaciones (Nielsen, 1993).

Mantenimiento: Plan para actualizaciones periódicas de modelos y mejoras basadas en retroalimentación (Sommerville, 2016).

3.2. Gestión de Requisitos

3.2.1. Identificación de Stakeholders

Como mencionan Leffingwell y Widrig (2003), "la identificación y gestión adecuada de los stakeholders es un factor clave para garantizar el éxito del proyecto" (p. 112).

Entre los principales interesados se encuentran:

1. Desarrolladores de modelos de Procesamiento de Lenguaje Natural (NLP).
2. Usuarios finales que requieren entender las decisiones tomadas por sistemas de inteligencia artificial, según Wiegers y Beatty (2013).
3. Auditores especializados en sistemas de IA, como lo destacan Selbst y Barocas (2018).
4. Reguladores encargados de supervisar la aplicación de estos modelos en áreas específicas, tal como señalan Jobin et al. (2019).

3.2.2. Requisitos Funcionales

Tabla *1.* *Matriz de Trazabilidad de Requisitos Funcionales (RF)*

Requisito Funcional (RF)	Objetivo	Componente	Prueba
RF1	Implementar técnicas de explicabilidad LIME y SHAP	Módulo de Explicabilidad (XAI)	Prueba unitaria de integración de LIME y SHAP
RF2	Visualizar la relevancia de términos en decisiones del modelo	Frontend (React)	Prueba de aceptación sobre la visualización de relevancia de términos
RF3	Permitir análisis de textos por parte del usuario	API Backend y Frontend (React)	Prueba de integración entre frontend y API para carga y análisis de textos
RF4	Almacenar históricos de análisis y explicaciones	Base de Datos (PostgreSQL, MongoDB)	Prueba de rendimiento en consultas de históricos de análisis
RF5	Permitir la comparación entre diferentes modelos y sus explicaciones	Componente de Comparación en Frontend	Prueba funcional de comparación entre modelos y resultados
RF6	Detectar y reportar posibles sesgos en decisiones del modelo	Módulo XAI y Backend	Prueba de precisión de la detección de sesgos en resultados de modelos

3.2.3. Requisitos No Funcionales

Tabla 2. Matriz de Trazabilidad de Requisitos No Funcionales (RNF)

Requisito No	Objetivo	Componente	Métrica de Evaluación

Funcional (RNF)			
RNF1	Generar explicaciones en un tiempo máximo de 5 segundos	API Backend y Módulo XAI	Tiempo de respuesta en carga de explicaciones
RNF2	Interfaz intuitiva y clara	Frontend (React)	Resultados de pruebas de usabilidad (cuestionarios de usabilidad)
RNF3	Manejar datos sensibles de forma segura	API Backend y Base de Datos (PostgreSQL, MongoDB)	Cumplimiento con normativas de seguridad de datos (GDPR, etc.)
RNF4	Escalabilidad para manejar múltiples solicitudes concurrentes	API Backend, Servidores (AWS/Google Cloud)	Pruebas de carga utilizando JMeter y análisis de rendimiento

3.3. Arquitectura Basada en Componentes (RUP)

3.3.1. Componente de Interfaz de Usuario (UI)

- Responsabilidad: Proporcionar formularios para entrada de textos y visualización de resultados (Tidwell, 2020).
- Tecnología: React.js con componentes reutilizables (Banks y Porcello, 2020).
- Interfaces: Se comunica con la API REST y con el componente de visualización.

3.3.2. Componente de Visualización

- Responsabilidad: Generar gráficos y representaciones visuales de las explicaciones (Munzner, 2014).
- Tecnología: Bibliotecas de visualización JavaScript (D3.js o similar) (Bostock et al., 2011).
- Interfaces: Recibe datos procesados de la API.

3.3.3. Componente API REST

- Responsabilidad: Gestionar solicitudes del frontend y coordinar el procesamiento (Richardson et al., 2013).

- Tecnología: Django REST Framework (Christie, 2022).
- Interfaces: Se comunica con todos los demás componentes.

3.3.4. Componente NLP

- Responsabilidad: Implementar modelos de procesamiento de lenguaje natural (BERT, GPT) (Devlin et al., 2019; Brown et al., 2020).
- Tecnología: Frameworks de ML como PyTorch o TensorFlow (Paszke et al., 2019).
- Interfaces: Recibe textos para análisis y devuelve resultados al módulo de explicabilidad.

3.3.5. Componente de Explicabilidad (XAI)

- Responsabilidad: Implementar técnicas LIME y SHAP para generar explicaciones (Arrieta et al., 2020).
- Tecnología: Bibliotecas específicas de XAI (Molnar, 2020).
- Interfaces: Recibe resultados del módulo NLP y genera explicaciones.

3.3.6. Componentes de Bases de Datos

- Responsabilidad: Almacenar datos estructurados (PostgreSQL) y no estructurados (MongoDB).
- Interfaces: Proporcionan servicios de persistencia a la API.

3.3.7. Comunicación entre Componentes

- Interfaces bien definidas mediante API REST para comunicación entre frontend y backend (Gamma et al., 1994).
- Uso de patrones de diseño como Observer para actualizaciones en tiempo real de visualizaciones.
- Implementación de mecanismos de caché para optimizar el rendimiento en consultas frecuentes.

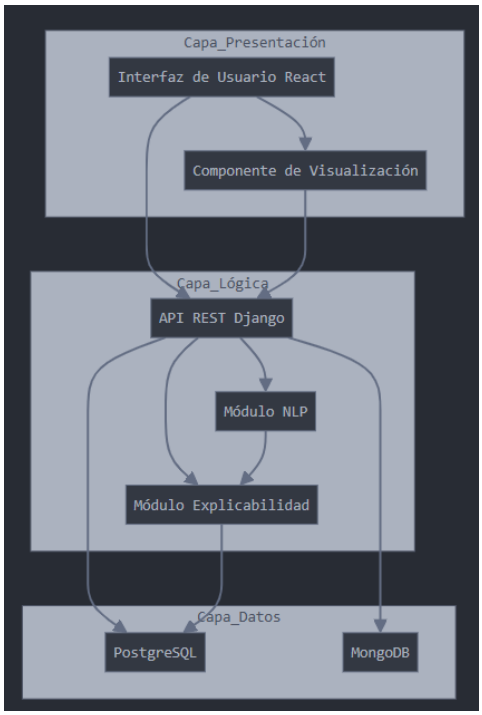

Figura 2. Arquitectura de Componentes del Sistema de Explicabilidad NLP

3.4. Verificación Continua (RUP)

3.4.1. Estrategia de Pruebas

- **Pruebas Unitarias**: Componentes individuales como NLP, XAI, API y UI.
- **Pruebas de Integración**: Validación de flujo de datos entre componentes.
- **Pruebas de Sistema**: Verificación de requisitos no funcionales como rendimiento y usabilidad.

3.4.2. Actividades de Aseguramiento de Calidad

- **Revisiones de Código**: Estándares como PEP 8 para Python y ESLint para JavaScript.
- **Auditorías de Explicabilidad**: Verificación de precisión y comprensibilidad de las explicaciones generadas.
- **Validación con Usuarios**: Pruebas de usabilidad mediante sesiones de feedback.

4. Conclusiones

En resumen, aplicar RUP en este proyecto permite una estructuración clara del desarrollo, asegurando que cada fase se cumpla de manera organizada y eficiente. La metodología garantiza la calidad del código, la integración efectiva de los modelos de IA y la generación de explicaciones comprensibles para mejorar la transparencia en el procesamiento de lenguaje natural.

Con respecto a lo establecido e investigado la explicabilidad en modelos de NLP con el uso de LIME y SHAP, no solo mejora la comprensión de los respectivos resultados, si no que hace referencia a la fomentación de la confianza de los clientes y de igual forma la adopción de la IA en Apps sensibles, en este aspecto se comprende a la toma de decisiones automatizadas, al asegurar la trasparencia y reducir los diferentes sesgos que se presentan.

5. Referencias bibliográficas

Martínez, A., & Martínez, R. (2014). Guía a rational unified process. *Escuela Politécnica Superior de Albacete–Universidad de Castilla la Mancha*.

Metzner, C., & Niño, N. (2016). El proceso de desarrollo RUP-GDIS. *Revista Venezolana de Computación*, 3(1), 13-22.

Ambler, S. W. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. John Wiley & Sons.

Apache Foundation. (2021). Apache JMeter. https://jmeter.apache.org/

Arlow, J., & Neustadt, I. (2005). UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd ed.). Addison-Wesley.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable

Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 58, 82-115.

Banks, A., & Porcello, E. (2020). Learning React: Functional Web Development with React and Redux (2nd ed.). O'Reilly Media.

Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice (2nd ed.). Addison-Wesley.

Beck, K. (2002). Test-Driven Development: By Example. Addison-Wesley.

Boehm, B. W. (1991). Software risk management: principles and practices. IEEE Software, 8(1), 32-41.

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D3: Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics, 17(12), 2301-2309.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.

Calero Sánchez, M., Gónzales Gónzales, J., Sánchez Berriel, I., Burrillo-Putze, G., & Roda García, J. (2024). El Procesamiento de Lenguaje Natural en la revisión de literatura científica. doi:10.55633/s3me/REUE030.2024

Cheng, H. F., Wang, R., Zhang, Z., O'Connell, F., Gray, T., Harper, F. M., & Zhu, H. (2019). Explaining decision-making algorithms through UI: Strategies to help non-expert stakeholders. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 1-12.

Christie, T. (2022). Django REST Framework. https://www.django-rest-framework.org/

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., & Stafford, J. (2010). Documenting Software Architectures: Views and Beyond (2nd ed.). Addison-Wesley.

Cohn, M. (2009). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley.

Deepseek. (2024, January 26). DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence. https://arxiv.org/pdf/2401.14196

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT 2019, 4171-4186.

Django. (2025). Django hace más fácil construir mejores aplicaciones web más rápidamente y con menos código. https://www.djangoproject.com/

Doshi-Velez, F., & Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. arXiv preprint arXiv:1702.08608.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. IBM Systems Journal, 15(3), 182-211.

Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley.