**CAPÍTULO 2: PROPUESTA DE SOLUCIÓN**

**2.1 Introducción al capítulo**

Este capítulo presenta la estrategia integral adoptada para el desarrollo del sistema de pruebas automatizadas del módulo Carga y Descarga del sistema de planificación de recursos de Habana Club International. La propuesta de solución que se expone a continuación surge como respuesta a las necesidades identificadas en el contexto empresarial, donde la ausencia de metodologías sistemáticas de pruebas automatizadas ha generado ineficiencias operativas y riesgos potenciales para la calidad del producto final.

El enfoque adoptado integra aspectos metodológicos y técnicos para crear un marco de trabajo que garantice la calidad del software a través de pruebas exhaustivas, reproducibles y mantenibles. La solución propuesta no se limita a la implementación de herramientas tecnológicas, sino que contempla un cambio en la filosofía de desarrollo, donde la calidad se construye desde las primeras etapas del ciclo de vida del software.

En las siguientes secciones se detallan los componentes fundamentales de la solución, partiendo del análisis del módulo Carga-Descarga, la metodología para el diseño de casos de prueba, la arquitectura de la solución de automatización, hasta las estrategias de gestión de datos e integración con el proceso de desarrollo. Cada elemento ha sido cuidadosamente considerado para asegurar una implementación exitosa que agregue valor significativo al proceso de desarrollo de software en AVANGENIO.

**2.2 Análisis del módulo "Carga-Descarga"**

El módulo "Carga-Descarga" constituye un componente crítico dentro del sistema de planificación de recursos de Habana Club International, encargado de gestionar los procesos logísticos relacionados con la entrada y salida de camiones, el control de peso y la coordinación de operaciones de movimiento de materiales.

**2.2.1 Caracterización funcional del módulo**

El análisis funcional del módulo revela una estructura compuesta por diversos submódulos interrelacionados, entre los que destacan:

1. **Gestión de tanques**: Encargado del seguimiento y control de las capacidades de almacenamiento y niveles de ocupación.
2. **Registro de vehículos**: Administra la información relacionada con las chapas (matrículas) de los vehículos que interactúan con el sistema logístico.
3. **Gestión de choferes**: Mantiene un registro actualizado de los conductores autorizados para las operaciones de carga y descarga.
4. **Laboratorio de calidad**: Integra los procedimientos de control de calidad de los productos durante los procesos de carga y descarga.
5. **Operaciones de carga**: Coordina todas las actividades relacionadas con la salida de productos desde las instalaciones.
6. **Operaciones de descarga**: Gestiona los procesos de recepción de materiales y productos, incluyendo verificación, pesaje y almacenamiento.

Los sistemas de gestión logística requieren un enfoque particular de pruebas debido a la naturaleza crítica de sus operaciones y la necesidad de integración con sistemas externos. La complejidad inherente a estos sistemas demanda estrategias de prueba que contemplen tanto los aspectos funcionales como los no funcionales, especialmente aquellos relacionados con la consistencia de datos y la trazabilidad de operaciones. [11]

**2.2.2 Flujos principales de usuario**

Los flujos de usuario en el módulo "Carga-Descarga" están estrechamente vinculados a los roles y autorizaciones asignados a cada usuario dentro del sistema. El acceso a las diferentes secciones depende de permisos específicos, lo que genera múltiples caminos de interacción basados en responsabilidades operativas.

Los principales flujos identificados incluyen:

1. **Flujo de administración**: El rol de Administrador tiene acceso completo a todas las secciones del módulo, incluyendo Chofer, Vehículo, Laboratorio de Calidad, Tanque, Carga y Descarga. Este flujo comprende la supervisión general y la capacidad de intervenir en cualquier punto del proceso.
2. **Flujo de gestión técnica**: El Técnico del Puesto de Dirección puede acceder a la mayoría de las secciones, con la notable excepción de la pestaña "Carga a Fabricación". Este flujo se centra en el monitoreo y control de las operaciones logísticas desde una perspectiva de dirección operativa.
3. **Flujo de fabricación**: Este flujo involucra a múltiples roles del área de fabricación:
   * **Jefe de Fabricación**: Puede acceder a la mayoría de secciones excepto la pestaña "Puesto de Dirección".
   * **Tecnólogo de Fabricación**: Enfocado en aspectos técnicos del proceso productivo.
   * **Fabricante**: Centrado en operaciones directas de producción.
   * **Jefe de Turno de Fabricación**: Supervisa las operaciones de carga y descarga durante su turno específico.
   * **Especialista de Economía de Fabricación**: Accede a información relacionada con aspectos económicos de los procesos.
4. **Flujo de control de calidad**: Este flujo incluye roles específicos para la verificación de calidad:
   * **Jefe de Laboratorio**: Tiene acceso exclusivo a la sección "Laboratorio de Calidad" junto con el Administrador.
   * **Especialista de Laboratorio de Fabricación**: Encargado de realizar pruebas y verificaciones de calidad específicas para los procesos de fabricación.

Cada uno de estos flujos incorpora secuencias de acciones específicas que reflejan las responsabilidades operativas de cada rol. La complejidad aumenta debido a las interdependencias secuenciales, donde la acción de un usuario (por ejemplo, el Jefe de Laboratorio verificando la calidad de un material) puede generar un punto de espera que requiere la intervención de otro usuario (como el Técnico del Puesto de Dirección autorizando la continuación del proceso) para avanzar en el flujo logístico.

El sistema de autenticación con contraseñas estandarizadas (TestErp2024\*+) para todos los usuarios facilita las pruebas en entornos controlados, mientras que la estructura de correos electrónicos con el formato test.erp.hci+[rol]@avangenio.com permite una clara identificación de los diferentes roles durante las pruebas.

La estructura de permisos segmentados asegura que los usuarios solo puedan interactuar con las secciones relevantes para sus responsabilidades, lo que contribuye a la integridad del proceso logístico, pero añade complejidad a la hora de diseñar pruebas automatizadas comprehensivas que cubran todos los posibles caminos de usuario.

**2.2.3 Complejidades técnicas identificadas**

Durante el análisis se identificaron varios desafíos técnicos que impactan directamente en la estrategia de pruebas:

1. **Ausencia de sistemas externos de pesaje**: Actualmente no existe un sistema externo real para el pesaje, por lo que se utiliza un mecanismo de verificación manual (check "pesa rota") que debe ser contemplado en los escenarios de prueba.
2. **Dependencias entre roles en procesos secuenciales**: Los flujos de trabajo incluyen puntos de pausa donde se requiere la intervención de roles específicos para continuar con el proceso, lo que genera complejidades para la prueba automatizada.
3. **Validaciones complejas con dependencias de calendario**: Existen reglas de negocio que involucran múltiples variables para la toma de decisiones logísticas, incluyendo validaciones relacionadas con lotes que incorporan componentes de calendario juliano.

Estos factores de complejidad deben ser considerados explícitamente en el diseño de pruebas automatizadas para evitar falsos positivos o negativos durante la ejecución. [12]

**2.3 Metodología para el diseño de casos de prueba**

La metodología adoptada para el diseño de casos de prueba se fundamenta en prácticas ágiles y principios de Behavior-Driven Development (BDD), buscando un enfoque sistemático que garantice la cobertura funcional completa del módulo.

**2.3.1 Proceso de identificación y documentación**

Para el diseño de casos de prueba se implementó un proceso estructurado en tres fases principales:

1. **Fase de análisis preliminar**: Estudio de las historias de usuario y especificaciones funcionales para identificar escenarios de prueba relevantes. Esta etapa incluyó la creación de una matriz de historias de usuario en formato Excel, donde se documentaron los aspectos clave a verificar por cada funcionalidad.
2. **Fase de diseño de casos**: Transformación de los escenarios identificados en casos de prueba formales, siguiendo inicialmente el formato BDD y posteriormente evolucionando hacia la sintaxis Gherkin para aprovechar sus capacidades avanzadas.
3. **Fase de validación**: Revisión y aprobación de los casos de prueba diseñados, garantizando su alineación con los requerimientos funcionales y la completitud de los escenarios.

**2.3.2 Implementación del enfoque BDD y migración a Gherkin**

Inicialmente, los casos de prueba fueron documentados siguiendo la estructura básica de BDD (Given-When-Then). Este enfoque permitió una comunicación clara de los escenarios a probar; sin embargo, a medida que avanzó el proyecto, se identificaron limitaciones en términos de reusabilidad y organización de pasos comunes.

Se realizó una migración progresiva hacia la sintaxis Gherkin, lo que permitió incorporar características avanzadas como:

1. **Background**: Para definir pasos comunes que se ejecutan antes de cada escenario.
2. **Scenario Outline**: Para parametrizar pruebas con conjuntos diferentes de datos.

Esta evolución metodológica optimizó significativamente la estructura de los casos de prueba, reduciendo la duplicación y mejorando la mantenibilidad del conjunto de pruebas. [13]

**2.3.3 Organización y gestión con Squash TM**

La gestión de los casos de prueba se realizó a través de Squash TM, una herramienta especializada que permitió:

1. **Organización jerárquica**: Estructuración de casos de prueba según módulos y funcionalidades.
2. **Trazabilidad**: Vinculación directa entre requisitos funcionales y casos de prueba correspondientes.
3. **Control de versiones**: Gestión de cambios en los casos de prueba a lo largo del tiempo.
4. **Exportación automatizada**: Generación de archivos .feature en formato Gherkin para su posterior implementación.

Para mantener la sincronización entre los casos diseñados y su implementación, se estableció un sistema de referencia cruzada utilizando una matriz en Excel que incluía:

* Identificador de historia de usuario
* Enlace a la especificación en el sistema de gestión de requisitos
* Identificador de caso de prueba en Squash TM
* Estado de implementación
* Responsable de la implementación

Esta estrategia proporciona visibilidad y control sobre el proceso de diseño e implementación de casos. [14]

**2.4 Arquitectura de la solución de automatización**

La arquitectura de la solución de automatización fue diseñada siguiendo principios de modularidad, reutilización y mantenibilidad, adoptando patrones de diseño establecidos en la industria para pruebas automatizadas.

**2.4.1 Framework de automatización seleccionado**

Tras un análisis comparativo de diferentes herramientas, se seleccionó Behave como framework principal para la implementación de pruebas automatizadas. Esta elección se fundamentó en:

1. **Compatibilidad con BDD/Gherkin**: Soporte nativo para la sintaxis utilizada en la especificación de casos.
2. **Integración con Python**: Aprovechamiento del ecosistema Python para la implementación de los scripts de prueba.
3. **Extensibilidad**: Capacidad para integrar librerías adicionales según necesidades específicas.
4. **Compatibilidad con Playwright**: Facilidad de integración con la herramienta seleccionada para la interacción con interfaces web.

Behave representa una solución robusta para la implementación de pruebas BDD, proporcionando un equilibrio adecuado entre simplicidad de uso y capacidades avanzadas. [15]

Entiendo, voy a corregir la estructura para que solo los steps y los archivos .feature estén dentro de la carpeta tests, sin incluir los demás archivos que mencionaste:

**2.4.2 Implementación del patrón Page Object Model (POM)** La arquitectura de la solución se fundamenta en el patrón Page Object Model, una práctica recomendada en la industria que promueve la separación entre la lógica de pruebas y la interacción con la interfaz de usuario. Se implementó una estructura jerárquica de clases que representan las diferentes páginas y componentes del sistema [16]:

HCI-MANUFACTURING-QA-AUTOMATION/

├── .venv

├── config

├── docs

├── pages/

│ └── .gitkeep

├── tests/

│ ├── steps/

│ │ ├── common\_steps.py

│ │ ├── carga\_steps.py

│ │ ├── descarga\_steps.py

│ │ └── ...

│ ├── carga.feature

│ ├── descarga.feature

│ ├── tanques.feature

│ └── ...

├── \_\_init\_\_.py

├── conftest.py

├── \_init\_.py

├── .env.example

├── .gitignore

├── .python-version

├── package-lock.json

├── package.json

├── playwright.config.ts

├── README.md

├── requirements.txt

└── utils/

├── context\_utils.py

├── fixtures.py

└── ...

Este enfoque aporta múltiples beneficios:

1. **Abstracción de la UI**: Encapsula los detalles de la interfaz de usuario, facilitando el mantenimiento cuando ocurren cambios visuales.
2. **Reutilización de código**: Las acciones comunes se implementan una única vez en las clases correspondientes.
3. **Legibilidad mejorada**: Los scripts de prueba expresan intenciones de negocio en lugar de detalles técnicos.
4. **Mantenibilidad**: Los cambios en la interfaz afectan únicamente a las clases de páginas correspondientes.

**2.4.3 Gestión de contexto y estado**

Un aspecto crucial en la arquitectura implementada es la gestión eficiente del contexto y estado durante la ejecución de pruebas. Behave proporciona un mecanismo nativo (el objeto "context") que fue aprovechado para:

1. **Compartir información entre pasos**: Transmisión de datos generados en un paso para su verificación en pasos posteriores.
2. **Mantener referencias a objetos de página**: Acceso consistente a las instancias de Page Objects a lo largo de la ejecución.
3. **Almacenar configuraciones dinámicas**: Adaptación del comportamiento de las pruebas según el entorno de ejecución.

Estas capacidades son particularmente relevantes en un sistema como el módulo Carga-Descarga, donde existen dependencias secuenciales entre operaciones y donde el estado anterior influye en el comportamiento esperado de pasos subsecuentes.

**2.4.4 Mecanismos de fixture para precondiciones**

Para abordar las necesidades de preparación del entorno de pruebas, se implementaron fixtures utilizando las capacidades de hooks de Behave. Estos mecanismos permiten:

1. **Autenticación automática**: Establecimiento de sesiones autenticadas antes de cada escenario.
2. **Preparación de datos**: Creación de precondiciones necesarias para la ejecución de pruebas.
3. **Limpieza posterior**: Restauración del estado inicial del sistema tras la ejecución.

Este enfoque de "preparación automática" reduce significativamente la duplicación de código y mejora la confiabilidad de las pruebas al garantizar un entorno consistente para cada ejecución. [17]

**2.5 Estrategia de parametrización y gestión de datos**

La gestión eficiente de datos de prueba constituye un factor crítico para el éxito de cualquier iniciativa de automatización. La estrategia adoptada se fundamenta en principios de separación de datos y lógica, configurabilidad y reutilización.

**2.5.1 Fuentes de datos para las pruebas**

La estrategia de gestión de datos para las pruebas automatizadas del módulo "Carga-Descarga" se fundamenta en un enfoque generativo y paramétrico, donde los propios especialistas en QA diseñan y proporcionan los conjuntos de datos necesarios para cubrir exhaustivamente los escenarios de prueba.

Las principales fuentes de datos utilizadas incluyen:

1. **Datos parametrizados mediante Gherkin**: La implementación de la sección "Examples" en los archivos .feature permite definir conjuntos de valores para variables específicas (como <nombre>, <matrícula>, etc.) que serán utilizados durante la ejecución de los escenarios. Esta técnica facilita la ejecución de un mismo escenario con múltiples combinaciones de datos sin duplicar código.
2. **Datos de configuración estáticos**: Información relativamente estable como credenciales de acceso, URLs de entorno, tiempos de espera y otros parámetros de configuración se almacenan en archivos de configuración separados, facilitando su modificación sin alterar el código de las pruebas.
3. **Datos dinámicos capturados durante la ejecución**: En escenarios donde ciertos valores son generados por el sistema (como identificadores únicos, timestamps o resultados de cálculos), estos son capturados durante la ejecución de un paso y almacenados en el contexto para su uso en pasos posteriores.
4. **Datos generados programáticamente (planificación futura)**: El Project Manager junto con el Jefe de los QA automatizados han propuesto implementar funciones generadoras que producirían datos válidos o inválidos según los requisitos del caso de prueba, especialmente para escenarios que requieren grandes volúmenes de datos o valores específicos como fechas en formato juliano para validaciones de lotes. Sin embargo, esta iniciativa aún se encuentra en fase de evaluación y no se ha confirmado su implementación definitiva.

El diseño de estos conjuntos de datos se basa en un análisis exhaustivo de las historias de usuario, buscando cubrir tanto escenarios típicos como casos límite y situaciones excepcionales que podrían ocurrir durante la operación real del sistema.

**2.5.2 Estrategia de aislamiento y repetibilidad**

Para garantizar la confiabilidad y repetibilidad de las pruebas automatizadas, se ha implementado una estrategia de aislamiento que minimiza las dependencias entre diferentes ejecuciones de prueba y asegura un entorno consistente para cada ciclo de evaluación.

Los componentes clave de esta estrategia incluyen:

1. **Hooks de preparación y limpieza**: Mediante la utilización del archivo environment.py de Behave, se han implementado mecanismos que ejecutan acciones específicas en momentos críticos del ciclo de vida de las pruebas:
   * **before\_all**: Configuración global única al inicio de toda la ejecución
   * **before\_feature**: Preparación específica para cada archivo .feature
   * **before\_scenario**: Configuración del entorno para cada escenario individual
   * **after\_scenario**: Limpieza y restauración del estado después de cada escenario
   * **after\_feature**: Acciones finales tras completar todos los escenarios de un feature
   * **after\_all**: Limpieza general al finalizar toda la ejecución
2. **Datos independientes por escenario**: Cada escenario de prueba opera con datos específicos y aislados, evitando interdependencias que podrían comprometer la repetibilidad de las pruebas. Esto se logra mediante la parametrización explícita y la regeneración de datos para cada ejecución.
3. **Restauración de estado**: Para escenarios que modifican el estado del sistema (como la creación de registros o la actualización de información existente), se implementan mecanismos automáticos que devuelven el sistema a un estado conocido tras la ejecución, ya sea mediante operaciones inversas o reinicialización de datos.
4. **Manejo de precondiciones**: Los requisitos previos para cada escenario se gestionan mediante pasos de configuración explícitos o a través de la sección "Background" en los archivos .feature, asegurando que todos los componentes necesarios estén disponibles antes de la ejecución de los pasos de prueba.

Esta estrategia de aislamiento no solo mejora la confiabilidad de los resultados de las pruebas, sino que también facilita la depuración al reducir las posibles causas de fallos intermitentes relacionados con estados inconsistentes del entorno de pruebas.

**2.6 Integración con el proceso de desarrollo**

La integración efectiva de las pruebas automatizadas en el flujo de desarrollo representa un factor determinante para maximizar su valor y garantizar su adopción por parte del equipo.

**2.6.1 Workflow de implementación**

Se estableció un workflow estructurado para la integración de nuevos casos de prueba:

1. **Diseño en Squash TM**: Creación y documentación inicial del caso de prueba siguiendo la sintaxis BDD/Gherkin.
2. **Revisión y aprobación**: Validación del caso de prueba por parte de analistas funcionales para garantizar su alineación con los requisitos.
3. **Exportación a archivo .feature**: Generación del archivo correspondiente en la estructura del proyecto de automatización.
4. **Verificación de disponibilidad de Page Objects**: Análisis de la necesidad de crear nuevas clases de página o reutilizar existentes.
5. **Implementación de steps**: Desarrollo de los pasos específicos requeridos por el escenario.
6. **Verificación inicial**: Ejecución local para confirmar el funcionamiento correcto.
7. **Integración en suite de pruebas**: Incorporación al conjunto completo de pruebas automatizadas.

Este proceso secuencial garantiza la trazabilidad completa desde el requisito funcional hasta la implementación automatizada, facilitando el mantenimiento y evolución del sistema de pruebas.

**2.6.2 Integración con CI/CD**

Aunque aún en fase inicial, se ha diseñado un plan para la integración del sistema de pruebas con la infraestructura de CI/CD, contemplando:

1. **Ejecución automática**: Configuración para la ejecución de pruebas tras cada integración de código.
2. **Reporting automatizado**: Generación de informes de resultados en formato estandarizado para facilitar su interpretación.
3. **Gestión de fallos**: Estrategia para la notificación y gestión de casos fallidos, incluyendo mecanismos de reintentos para identificar fallos intermitentes.
4. **Análisis de tendencias**: Almacenamiento histórico de resultados para identificar patrones de comportamiento y áreas problemáticas.

Este enfoque, alineado con las prácticas recomendadas por Humble y Farley [19], busca maximizar el valor de las pruebas automatizadas incorporándolas como elemento fundamental del pipeline de entrega continua.

**2.7 Implementación de mantenimiento asistido por IA**

Como elemento innovador dentro de la propuesta, se ha incorporado un enfoque de mantenimiento asistido por inteligencia artificial para los scripts de prueba automatizados, lo que representa un avance significativo en la sostenibilidad del proyecto a largo plazo.

**2.7.1 Asistencia de Claude para actualización de código**

Una de las principales desventajas de las pruebas automatizadas tradicionales es su fragilidad ante cambios en la estructura de la aplicación. Para mitigar este problema, hemos implementado un flujo de trabajo de mantenimiento asistido por IA que:

1. **Proporciona actualizaciones contextuales**: Cuando se producen cambios en el proyecto posteriores a la implementación inicial, proporcionamos a Claude el código existente junto con información sobre los cambios necesarios.
2. **Genera código actualizado con coherencia estructural**: Claude analiza la estructura actual del proyecto, comprende las modificaciones requeridas y propone implementaciones que mantienen la coherencia con los patrones y estándares establecidos.
3. **Preserva la lógica de negocio**: Al actualizar selectores o flujos de interacción, Claude mantiene intacta la lógica fundamental de las pruebas, enfocándose únicamente en los aspectos técnicos que requieren actualización.

Según experiencias documentadas por Alshahwan et al. [20], este enfoque puede reducir hasta en un 60% el tiempo dedicado al mantenimiento de pruebas automatizadas.

**2.7.2 Beneficios del mantenimiento asistido por IA**

La implementación de este enfoque ofrece ventajas significativas:

1. **Respuesta ágil ante cambios**: Capacidad para adaptar rápidamente los scripts de prueba cuando la interfaz o comportamiento de la aplicación evoluciona.
2. **Consistencia en la actualización**: Garantía de que las modificaciones siguen patrones coherentes en todo el código base, evitando soluciones ad-hoc que puedan introducir inconsistencias.
3. **Transferencia de conocimiento**: Claude comprende la estructura y patrones del proyecto, lo que facilita la incorporación de nuevos miembros al equipo, quienes pueden interactuar con la IA para entender y modificar el código existente.

Este enfoque representa una dirección prometedora para organizaciones que buscan mantener un equilibrio entre la cobertura de pruebas automatizadas y los recursos necesarios para su mantenimiento, permitiendo que el equipo se concentre en actividades de mayor valor añadido.

**2.8 Conclusiones del capítulo**

La propuesta de solución presentada en este capítulo constituye un enfoque integral para la automatización de pruebas del módulo Carga-Descarga, abordando desde aspectos metodológicos hasta implementaciones técnicas específicas.

Los elementos clave que definen esta propuesta incluyen:

1. **Enfoque BDD/Gherkin**: Adopción de un paradigma que facilita la comprensión de las pruebas por parte de todos los interesados, mejorando la comunicación entre equipos técnicos y de negocio.
2. **Arquitectura POM**: Implementación de una estructura modular basada en Page Objects que garantiza la mantenibilidad y escalabilidad de la solución.
3. **Gestión estructurada**: Establecimiento de procesos formales para el diseño, implementación y mantenimiento de casos de prueba.
4. **Exploración de innovaciones**: Apertura hacia tecnologías emergentes que pueden optimizar el costo-beneficio de la automatización.

La validez de esta propuesta se fundamenta en su alineación con las mejores prácticas de la industria, como lo demuestran las referencias a trabajos de autores reconocidos en el campo del testing automatizado. Adicionalmente, el diseño responde directamente a las necesidades y complejidades específicas identificadas en el análisis del módulo Carga-Descarga.

La implementación práctica de esta solución, que se detallará en el siguiente capítulo, constituirá una prueba empírica de la efectividad del enfoque propuesto y proporcionará datos cuantitativos sobre sus beneficios en términos de eficiencia, cobertura y capacidad de detección de defectos.

**Referencias Bibliográficas adicionales para el Capítulo 2**

1. Sommerville, I. (2015). *Software Engineering (10th Edition)*. Pearson. <https://dn790001.ca.archive.org/0/items/bme-vik-konyvek/Software%20Engineering%20-%20Ian%20Sommerville.pdf>​[dn790001.ca.archive.org](https://dn790001.ca.archive.org/0/items/bme-vik-konyvek/Software%20Engineering%20-%20Ian%20Sommerville.pdf?utm_source=chatgpt.com)
2. Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional. https://www.mountaingoatsoftware.com/books/succeeding-with-agile-software-development-using-scrum
3. Smart, J. F. (2021). *BDD in Action: Behavior-Driven Development for the whole software lifecycle (2nd Edition)*. Manning Publications. https://www.manning.com/books/bdd-in-action-second-edition
4. Gregory, J., & Crispin, L. (2014). *More Agile Testing: Learning Journeys for the Whole Team*. Addison-Wesley Professional. https://www.informit.com/store/more-agile-testing-learning-journeys-for-the-whole-9780321967053
5. Hellmann, D. (2017). *The Hitchhiker's Guide to Python: Best Practices for Development*. O'Reilly Media. https://docs.python-guide.org/
6. Fowler, M. (2013). *PageObject*. Martin Fowler's Blog. https://martinfowler.com/bliki/PageObject.html
7. Percival, H. (2017). *Test-Driven Development with Python (2nd Edition)*. O'Reilly Media. https://www.obeythetestinggoat.com/
8. Kaner, C., Bach, J., & Pettichord, B. (2008). *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley. https://www.wiley.com/en-us/Lessons+Learned+in+Software+Testing%3A+A+Context+Driven+Approach-p-9780471081128
9. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional. https://continuousdelivery.com/
10. Leotta, M., Clerissi, D., Ricca, F., & Spadaro, C. (2020). *Improving Test Suite Maintainability with the Page Object Pattern: An Industrial Case Study*. Software Testing, Verification and Reliability, 30(1), e1698. <https://sepl.dibris.unige.it/publications/2013-leotta-ICSTW.pdf>