

Evaluación de una Guía Metodológica para el Desarrollo de Software Científico Con Estándares de Calidad

Leonard Adrián Gómez García

Investigación para optar al título de Astrónomo

Asesor de Trabajo de Grado
Luis Fernando Quiroga Peláez, PhD.

Universidad de Antioquia
Facultad de Ciencias Exactas y Naturales, Instituto de Física
Pregrado UdeA
Medellín, Antioquia
2025

Cita	Gómez y Quiroga, 2025 [1]
Referencia Estilo IEEE (2020)	[1] Gómez, L. A., y Quiroga, L. F. "Evaluación de una Guía Metodológica para el Desarrollo de Software Científico Con Estándares de Calidad", Investigación, Pregrado UdeA, Universidad de Antioquia, Medellín, 2025.





Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano/Director Adriana Echavarría Isaza.

Jefe departamento: Jorge Mario Osorio Guillen.

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

Dedicatoria

A mis padres, por ser el ejemplo vivo de amor incondicional y esfuerzo; su invaluable apoyo hizo posible la realización de este trabajo.

Agradecimientos

Agradezco profundamente al Prof. Luis Quiroga por su guía rigurosa, sus observaciones críticas y especialmente por su confianza a lo largo de este proyecto. Extiendo también mi gratitud a las y los estudiantes que participaron en las distintas fases del desarrollo; su trabajo y compromiso fueron esenciales para alcanzar los resultados, manteniendo siempre su identidad en reserva conforme a los lineamientos del estudio.

TABLA DE CONTENIDO

RESUMEN 6
ABSTRACT 7
I. INTRODUCCIÓN 8
II. ESTADO DEL ARTE 9
III. MARCO TEÓRICO 13
IV. OBJETIVOS 21
V. PROBLEMA DE INVESTIGACIÓN 22
VI. METODOLOGÍA 23
VII. RESULTADOS 32
VIII. DISCUSIÓN 43
IX. RECOMENDACIONES 44
X. CONCLUSIONES 45
REFERENCIAS 47
ANEXOS 51

LISTA DE TABLAS

Tabla I Resultados de evaluación por criterios — Software Comparativo de Métodos Numéricos 35
Tabla II Resultados de evaluación por criterios - Simulador SPH. 37
Tabla III Horas por grupo (descriptivos sobre horas totales por estudiante). . . . 39
Tabla IV Distribución de tiempo por tipo de actividad (porcentaje). 39
Tabla V Rúbrica técnica comparativa. 40
Tabla VI Percepción de la guía (media 1–5; $n = 3$). 41
Tabla VII Enlaces a entregables. 51

RESUMEN

El software científico es central en la investigación actual, en especial en las ciencias exactas, donde las simulaciones y los métodos numéricos permiten modelar y analizar fenómenos complejos de difícil acceso experimental [1]. Persisten, sin embargo, retos de eficiencia, mantenibilidad, reproducibilidad y calidad del código producido por científicos sin formación formal en ingeniería de software [2]; además, la predicción y gestión de la mantenibilidad —clave para la evolución y sostenibilidad— sigue siendo un desafío metodológico y técnico [3].

Este trabajo propone el uso de una guía metodológica para el desarrollo de software científico orientada a que los académicos produzcan artefactos de alta calidad, evaluada mediante un estudio comparativo en un entorno educativo. Como caso de estudio, se diseñó un software para el análisis de estabilidad de discos galácticos en simulaciones, ejecutado por estudiantes de Astronomía de la Universidad de Antioquia distribuidos en dos grupos: uno que sigue la guía y otro sin ella. Adicionalmente, se incluyen dos casos externos: la implementación de un método numérico desarrollado por el autor de este trabajo y un simulador de fluidos usando la técnica SPH (Smoothed Particle Hydrodynamics) elaborado por un estudiante avanzado de Astronomía de la Universidad de Antioquia. La calidad se evalúa con criterios técnicos, estructurales y procesuales: calidad del código, rendimiento, tiempo de desarrollo y percepción de los participantes.

Se presentan resultados parciales que indican que el uso de la guía reduce el tiempo de desarrollo y mejora criterios de calidad del software final, evidenciando un efecto positivo en el proceso de construcción de software científico en contextos formativos y de investigación.

Palabras clave **— Software científico, Buenas prácticas de programación, Guía metodológica**

ABSTRACT

Scientific software plays a central role in contemporary research, especially in the exact sciences, where simulations enable the modeling and study of complex phenomena that are difficult to access experimentally [1]. Nevertheless, challenges persist regarding the efficiency, maintainability, reproducibility, and quality of code produced by scientists without formal training in software engineering [2]; moreover, the prediction and management of maintainability—key to software evolution and sustainability—remain methodological and technical hurdles [3].

This work proposes a methodological guide for the development of scientific software aimed at enabling academics to produce high-quality artifacts, evaluated through a comparative study in an educational setting. As a case study, we designed software project for analyzing the stability of galactic disks in simulations, carried out by Astronomy students at the University of Antioquia, divided into two groups: one following the guide and another without it. In addition, two external cases are included: a numerical simulator developed by the author of this work and an SPH (Smoothed Particle Hydrodynamics) simulator developed by a final semester Astronomy student at the University of Antioquia. Quality is assessed using technical, structural, and process criteria: code quality, software performance, development time, and participants' perceptions.

We present preliminar results indicating that using the guide reduces development time and improves quality criteria of the students' final software, showing a positive effect on the process of building scientific software in educational and research contexts.

Keywords **— Scientific software, Best programming practices, Methodological guide**

I. INTRODUCCIÓN

La producción de nuevo conocimiento en ciencias físicas y, en particular, en astrofísica, depende cada vez más de la capacidad de diseñar, implementar y mantener código fuente que sirva como instrumento experimental [2]. A diferencia del software comercial, cuyo ciclo de vida obedece a presiones de mercado, el software científico se desarrolla en entornos de alta incertidumbre, con plazos de publicación y la necesidad de reproducir resultados.

En la última década se han propuesto lineamientos que trasladan principios de la ingeniería de software —control de versiones, pruebas automatizadas y documentación viva— al laboratorio computacional [4]. Sin embargo, la implementación efectiva de estos lineamientos en contextos formativos sigue siendo limitada. Diversos estudios señalan que los estudiantes priorizan "hacer que funcione" sobre la mantenibilidad y la trazabilidad del código, lo que merma la credibilidad de los hallazgos numéricos, hábito que se traslada a su ejercicio profesional una vez obtienen su título [5].

En respuesta, este trabajo evalúa empíricamente el impacto de una Guía Instructiva y Metodológica para el Desarrollo de Software Científico en el rendimiento de proyectos de simulación de estabilidad de discos galácticos desarrollados por estudiantes universitarios. Con un diseño cuasi–experimental, se comparan dos cohortes: una que adopta la guía (grupo experimental) y otra que trabaja sin ella (grupo control). El propósito es determinar si la intervención metodológica mejora la eficiencia del proceso y la calidad de los entregables sin obstaculizar la creatividad investigativa.

II. ESTADO DEL ARTE

El desarrollo de software ha tenido un rol crucial en el campo científico y la producción de conocimientos del mismo en diferentes áreas del conocimiento. El desarrollo y uso de estas herramientas ha resultado indispensable, por ejemplo, en la geofísica, física estadística y astrofísica -abordada en este proyecto- debido al gran volumen de datos a utilizar, el aumento en la complejidad de los modelos aplicados en las diferentes áreas de investigación de frontera y la necesidad de realizar simulaciones predictivas de los fenómenos a estudiar. En consecuencia, las metodologías de desarrollo de estas herramientas y productos de software suponen también un foco de atención respondiendo a las necesidades intrínsecas de la ciencia, la transparencia del acto de la investigación y la reproducibilidad de los experimentos. Como señala Storer [6], existen brechas en la programación científica y la ingeniería de software, pero la adopción de buenas prácticas -control de versiones, pruebas, revisión de código y documentación- es clave para mejorar la mantenibilidad, eficiencia y confiabilidad del software de investigación.

Debido a la importancia de asegurar estas necesidades, han surgido diferentes perspectivas y posturas respecto a temas como las prácticas de producción de software científico, cómo desarrollarlo, y qué metodologías del desarrollo de software comercial pueden adoptarse y adaptarse en el contexto científico para garantizar la eficiencia, mantenimiento y escalabilidad de los sistemas, conservando su validez académica.[7].

A. Buenas prácticas como base del desarrollo eficiente

Una de las corrientes que abordan esta problemática propone aplicar principios formales de ingeniería de software al entorno científico. Wilson et al.[2] argumentan que la falta de formación en metodologías de desarrollo de software en ambientes científicos genera un conjunto de herramientas, aplicaciones y sistemas difíciles de mantener, verificar, reproducir y escalar. En su trabajo de buenas prácticas para la producción de software científico, destacan el uso de control de versiones, pruebas automatizadas, revisión por pares del código y documentación clara como factores clave con el objetivo de mejorar la calidad del software

y, por lo tanto, de la actividad científica.

Complementando lo mencionado previamente, Hannay [4] plantea la necesidad de un mayor enfoque en la formación de desarrollo de software que reciben los científicos, al sugerir que un entrenamiento básico en el mismo podría no proveer las suficientes herramientas para desarrollar productos de software más robustos en un área de proyectos siempre crecientes.

B. Críticas a la aplicabilidad directa de estándares industriales

En contraposición, algunos autores han cuestionado la aplicabilidad directa de las buenas prácticas de desarrollo de software en contextos científicos. Johanson [8] sugiere que la programación científica suele adoptar un carácter experimental. Esto se debe a que, si bien el científico puede tener claridad sobre los fundamentos teóricos que serán utilizados en un proyecto de software, la forma en que este conocimiento será implementado puede no ser clara desde el comienzo.

Como en la ciencia no suele haber un camino predefinido, sino que este se forma a través del análisis de resultados, es igualmente necesario evaluar los resultados de una versión de software antes de avanzar con la siguiente. El autor define la naturaleza de esta dinámica como "iterativa", y, en parte, es precisamente esta condición la que lleva a que los científicos rara vez consideren una estructura predefinida para sus desarrollos. A esto se suma el hecho de que estos proyectos, generalmente pequeños al comienzo, tienden a crecer conforme reciben retroalimentación de la comunidad. Como resultado, los requerimientos suelen formularse en fases avanzadas de desarrollo, cuando el software ya está casi terminado.

Otro cuestionamiento sobre la aplicabilidad de estas prácticas surge de los procesos de validación y verificación. En el ámbito de las pruebas, la verificación se refiere a demostrar que la implementación de los algoritmos y ecuaciones utilizados en el software es correcta, mientras que la validación consiste en confirmar que los resultados generados por el sistema sean coherentes con las observaciones empíricas. La validación, en particular, representa un desafío considerable en este tipo de desarrollos, dado que los objetos de estudio suelen presentar dificultades asociadas a su tamaño, complejidad, peligrosidad o alto costo de análisis.

C. Transparencia, apertura y reproducibilidad

Un tercer eje de análisis aborda la necesidad de apertura y auditabilidad del software. Ince, Hatton y Graham-Cumming [5] señalan que la falta de acceso al código fuente limita la posibilidad de verificar y reproducir los resultados obtenidos mediante simulaciones o modelos computacionales. Plantean que la ciencia basada en software debe alinearse con los principios de ciencia abierta: compartir el código es tan esencial como compartir los datos. Este enfoque no solo mejora la reproducibilidad, sino que también fomenta la colaboración y evita la duplicación de esfuerzos.

Asimismo, en una observación sobre las prácticas de adopción de herramientas de software dentro de la comunidad científica, Joppa et al. [9] reportaron que tan solo un 8 % de los investigadores afirmó haber elegido un software principalmente por haberlo validado frente a otros métodos. En contraste, otros factores reportados incluyeron el uso de herramientas con interfaces sencillas o "click-and-run" (57 %), la recomendación de un colega cercano (18 %), una recomendación personal (9 %) o la reputación del desarrollador (7 %).

Cabe señalar que estos porcentajes no representan categorías mutuamente excluyentes y pueden derivarse de subgrupos dentro de la muestra total, o de respuestas múltiples por parte de los encuestados. Se destaca la consideración de que algunos investigadores utilizan software como una "caja negra": sin comprender realmente su funcionamiento interno. Esto no solo representa un riesgo epistemológico, sino que pone en duda la validez de los resultados científicos; la popularidad de un software no garantiza rigurosidad, y en la facilidad de uso el investigador podría ignorar detalles operacionales, comprometiendo la calidad y validez de sus resultados. Como resultado, se propone la implementación de un estándar que garantice la transparencia e inteligibilidad del código, asegurando la reproducibilidad mediante la simulación y la confiabilidad a través de la revisión científica por pares. Adicionalmente, las revisiones de software en la ciencia, concluye Rokem [10], pueden incluso evitar malgastos que surgirían de permitir que los errores en el código alcancen una etapa avanzada de los proyectos.

Se sugiere, además, tomar como referencia medios científicos de difusión que ya em-

plean filtros similares, y de los cuales este estándar podría beneficiarse o incluso componerse. Entre ellos destacan *IPOL: Image Processing On Line* (que exige el envío del código para revisión por pares), *Insight Journal* (centrado en la compilación automatizada y pruebas de código) y el *Journal of Open Research Software* (que también revisa el código y publica descripciones concisas del software utilizado en cada investigación).

Según estos autores, la formación en principios básicos de desarrollo, junto con la documentación adecuada, debería ser parte de la formación de cualquier científico que trabaje con herramientas computacionales.

En conjunto, los diferentes enfoques expuestos evidencian que el desarrollo de software científico se encuentra en una intersección compleja entre las exigencias de rigor metodológico y la realidad dinámica de la investigación académica. Si bien existen consensos sobre la necesidad de mejorar la calidad del software mediante prácticas formales, también se reconoce la importancia de adaptar dichas prácticas al contexto particular de la ciencia. La apertura del código y la comprensión crítica de las herramientas utilizadas emergen como elementos clave para avanzar hacia una ciencia más reproducible, colaborativa y confiable[11].

El presente trabajo se enmarca precisamente en esta línea: verificar la eficiencia de una guía de desarrollo de software científico a través de su aplicación comparativa en un entorno educativo, específicamente mediante el desarrollo de proyectos de software para el estudio de estabilidad de galaxias disco en simulaciones de formación y evolución de galaxias, realizado por estudiantes universitarios. Esto permitirá observar, de forma empírica, si una intervención metodológica como una guía puede mejorar aspectos clave del desarrollo del software (como la organización del código, la documentación y el rendimiento del producto final) sin obstaculizar el flujo natural del trabajo investigativo.

Con base en ello, el siguiente capítulo presentará los fundamentos conceptuales que sustentan esta investigación, definiendo con mayor precisión los elementos teóricos involucrados: desde la naturaleza del software científico y el concepto de eficiencia, hasta las nociones de verificación, reproducibilidad y simulación computacional.

III. MARCO TEÓRICO

A. Software científico

El término "software científico" hace referencia a aquellos programas informáticos diseñados y desarrollados específicamente para apoyar, automatizar o ejecutar procesos relacionados con la investigación científica. A diferencia del software comercial o de consumo, cuyo propósito es satisfacer necesidades de mercado, el software científico se orienta a resolver problemas complejos dentro de campos como la física, la biología, la química computacional, la geociencia o la astronomía.

Este tipo de software suele estar estrechamente vinculado a modelos matemáticos, simulaciones numéricas, procesamiento masivo de datos o análisis estadísticos avanzados. Con frecuencia, es desarrollado directamente por científicos o equipos de investigación, muchas veces sin una formación formal en ingeniería de software, lo que conduce a enfoques de programación altamente personalizados y con escasa estandarización. De acuerdo con Hannay [4], el software científico tiende a ser producido en entornos académicos bajo condiciones experimentales, lo que lo caracteriza por su flexibilidad, rápida evolución y dependencia del conocimiento disciplinar específico. Por su parte, Wilson et al. [2] advierten que la carencia de prácticas estructuradas en su desarrollo conlleva riesgos para la mantenibilidad, comprensión y reproducibilidad de los resultados científicos que dependen de estas herramientas.

Además, Johanson [8] sostiene que el desarrollo de software científico tuvo su origen en equipos pequeños de investigación, donde la coordinación interna podía mantenerse sin necesidad de aplicar principios formales de ingeniería de software. Sin embargo, debido a que estos proyectos suelen extenderse por años o incluso décadas, muchos de ellos crecieron en escala y pasaron a involucrar equipos interdisciplinarios. En este nuevo contexto, se hizo relevante corregir la falta de rigurosidad en el desarrollo inicial y adoptar así principios de ingeniería de software.

En este sentido, el software científico no debe evaluarse únicamente desde los criterios tradicionales de calidad del software, sino también desde su utilidad para la investigación, su capacidad para reflejar modelos teóricos y su contribución al conocimiento reproducible. Esta

particularidad justifica la necesidad de adoptar enfoques específicos, tales como guías metodológicas adaptadas a los contextos científicos, que equilibren calidad técnica y viabilidad investigativa.

B. Eficiencia en el desarrollo de software científico

El concepto de eficiencia en el desarrollo de software científico es multidimensional y depende del contexto en el que se lo aplique. A diferencia del software comercial, donde la eficiencia suele asociarse con métricas de mercado como el tiempo de entrega, la escalabilidad o el consumo de recursos, en el ámbito científico, la eficiencia incluye aspectos relacionados con la calidad de los resultados, la reproducibilidad de los experimentos computacionales y la sostenibilidad del código a largo plazo.

Desde una perspectiva tradicional, la eficiencia se puede entender como la relación entre el esfuerzo invertido en el desarrollo (tiempo, recursos, complejidad técnica) y la calidad o utilidad del producto obtenido (funcionalidad, mantenibilidad, rendimiento y documentación). Sin embargo, en entornos científicos este concepto puede ampliarse para incluir la confiabilidad del software en la producción de conocimiento, su capacidad de ser comprendido por otros investigadores, y su adaptabilidad a futuros experimentos o modificaciones teóricas, características claves propuestas en iniciativas como el proyecto Productivity and Sustainability Improvement Planning (PSIP), impulsado por el U.S. Department of Energy, como parte de sus estrategias para mejorar la eficiencia del desarrollo de software científico. [12, 13].

Según Wilson et al. [2], una de las principales barreras para lograr eficiencia en el desarrollo científico es la ausencia de estructuras de trabajo compartidas. El código suele ser escrito rápidamente, sin planificación previa, pruebas automatizadas ni documentación clara, lo que da lugar a productos difíciles de mantener y compartir. Además, el impacto de esta situación se amplifica si se tiene en cuenta que este tipo de software tiende a compartirse con otros científicos y a utilizarse en nuevos proyectos. En ese contexto, la eficiencia no solo se ve afectada en términos técnicos, sino también en términos epistemológicos: si los resultados obtenidos mediante simulaciones no pueden ser replicados o entendidos por terceros, su valor

científico se ve significativamente reducido.

Por otro lado, Segal [14], en su estudio etnográfico sobre desarrolladores de software científico, sostiene que la eficiencia no debe medirse únicamente por indicadores técnicos, sino también por la capacidad del software de acompañar el proceso de descubrimiento. En otras palabras, un software puede ser considerado eficiente si permite al científico iterar, experimentar y modificar el código con facilidad, aún si su estructura no es óptima según criterios formales de ingeniería.

En el presente marco de estudios comparativos, la eficiencia puede ser operacionalizada mediante estos indicadores [15]:

1. Tiempo de desarrollo: desde el diseño hasta la ejecución funcional del software.
2. Calidad estructural del código: claridad, modularidad, coherencia.
3. Rendimiento computacional: tiempos de simulación, consumo de recursos.
4. Calidad de la documentación: claridad, completitud, utilidad para otros desarrolladores.
5. Facilidad de mantenimiento: grado de dificultad para modificar o extender el software.
6. Satisfacción o percepción del desarrollador: mediante cuestionarios cualitativos (opcional).

A estos criterios pueden sumarse indicadores de reproducibilidad y transparencia, como sugiere Ince et al. [5], dado que la eficiencia está estrechamente ligada al impacto científico del software.

Finalmente, Heroux et al. [12] proponen un enfoque mixto para asegurar eficiencia en contextos científicos, adaptando el uso de buenas prácticas sin retrasar el avance científico de los investigadores.

C. Verificación, validación y reproducibilidad

En el desarrollo de software científico, los conceptos de verificación, validación y reproducibilidad son fundamentales para garantizar la confiabilidad tanto del software como

de los resultados científicos que se obtienen a partir de su uso. A continuación, se describen cada uno de estos conceptos.

1) Verificación: Es la evaluación técnica orientada a determinar si el software cumple con los objetivos para los cuales fue diseñado. Es decir, verifica que el código cumpla con los requisitos funcionales especificados y que no contenga errores lógicos o sintácticos. Según Pressman & Maxim [16], este proceso responde a la pregunta: "¿Estamos construyendo el producto correctamente?"

En software científico, la verificación implica que los algoritmos estén correctamente implementados, que las unidades de código funcionen como se espera (a menudo mediante pruebas unitarias) y que el software se ejecute sin fallos internos.

2) Validación: Busca responder a la pregunta: "¿Estamos construyendo el producto correcto?" Es decir, evalúa si el software cumple con el propósito científico para el cual fue creado. En este caso, no basta con que el software funcione, sino que debe reflejar adecuadamente el modelo científico o experimental. Esto puede implicar la comparación de los resultados con datos observacionales, modelos teóricos o con resultados obtenidos por otros métodos.

En contextos como el análisis de la estabilidad de discos galácticos, la validación podría, por ejemplo, verificar que los perfiles de densidad superficial y dispersión de velocidades sigan comportamientos plausibles desde el punto de vista astrofísico.

3) Reproducibilidad: Es la capacidad de un experimento, ensayo, software, entre otros, de ser reproducido o replicado por terceros bajo las mismas condiciones. En el caso de un experimento computacional, se considera reproducible cuando otro investigador puede obtener los mismos resultados utilizando el mismo código, los mismos datos y las mismas condiciones iniciales.

Ince, Hatton y Graham-Cumming [5] sostienen que, sin reproducibilidad, el conocimiento generado mediante software científico no puede considerarse completamente válido.

Este aspecto ha cobrado especial importancia en los últimos años, donde se han documentado numerosos casos de estudios cuyos resultados no pudieron ser replicados por falta de acceso al código fuente, mala documentación o dependencia de configuraciones no estandarizadas.

De acuerdo con Peng [17], la reproducibilidad debe considerarse como el mínimo estándar para la credibilidad de un resultado científico computacional. En este sentido, la calidad del software y su documentación afectan directamente a la calidad de la ciencia producida.

4) Interrelación y relevancia para el presente estudio: Estos tres conceptos no deben entenderse como procesos aislados, sino como dimensiones complementarias que permiten evaluar tanto el software como el proceso científico en su conjunto. En el contexto del presente trabajo, dichos principios orientan la construcción de criterios para valorar la eficiencia y la calidad en el desarrollo de proyectos de software para el análisis de galaxias de disco simuladas realizados por estudiantes, con y sin el apoyo de una guía metodológica. La presencia o ausencia de documentación clara, estructuras de código verificables y resultados válidos será analizada, entre otros aspectos, con base en estos principios.

D. Aprendizaje basado en proyectos en el desarrollo de software científico

El aprendizaje basado en proyectos (ABP) es una metodología pedagógica centrada en el estudiante, en la cual el conocimiento se construye mediante el diseño, la ejecución y la evaluación de un proyecto concreto, generalmente orientado a la solución de un problema real. Esta estrategia no solo promueve el dominio conceptual y técnico, sino también el desarrollo de competencias como la resolución de problemas, el trabajo colaborativo, la autonomía y la toma de decisiones.

Esta metodología ha demostrado ser especialmente eficaz en el ámbito de la enseñanza de la programación y el desarrollo de software científico: según Thomas [18] y Barron et al. [19], los estudiantes que aprenden mediante proyectos tienden a desarrollar una comprensión más profunda de los conceptos, ya que deben aplicarlos en contextos auténticos, reflexionar sobre sus elecciones y enfrentar las consecuencias de sus decisiones de diseño. Este enfoque

resulta particularmente adecuado cuando se busca evaluar la aplicabilidad de herramientas metodológicas o guías de desarrollo, tal como se propone en este estudio; al enfrentar a los estudiantes a la tarea de construir un software científico —en este caso, para el análisis de la estabilidad de galaxias de disco simuladas— se genera un entorno de aprendizaje en el que emergen de manera natural los desafíos relacionados con la estructuración del código, la planificación del desarrollo, la documentación y la validación de resultados.

Además, como señalan Capretz et al. [20], el ABP permite observar con mayor claridad cómo los estudiantes internalizan o ignoran las buenas prácticas, dependiendo de si tienen o no apoyo estructurado. Esto lo convierte en un contexto ideal para estudiar de manera comparativa el efecto de una guía de desarrollo.

Desde una perspectiva evaluativa, esta metodología también permite diseñar instrumentos objetivos y cualitativos para medir el desempeño técnico y la experiencia de aprendizaje, generando evidencia empírica sobre el impacto pedagógico de introducir marcos metodológicos formales en contextos formativos.

Por lo tanto, el uso del aprendizaje basado en proyectos no solo se justifica desde el punto de vista educativo, sino que también proporciona un contexto metodológico válido y controlado para estudiar el efecto real de una guía de desarrollo de software científico en un entorno con variables parcialmente controladas, pero representativas del trabajo colaborativo en ciencia.

E. Guía instructiva y metodológica para el desarrollo de software científico

La *Guía Instructiva y Metodológica para el Desarrollo de Software Científico* surge como un marco práctico orientado a disminuir la brecha entre la programación científica y la ingeniería de software a partir de una revisión sistemática de la literatura y el análisis de casos en física, astronomía y biología computacional; en el cual se identificaron barreras estructurales de tipo técnico, formativo y cultural que afectan la calidad, sostenibilidad y reproductibilidad del software [8, 1]. La ausencia de formación en buenas prácticas, el uso de leguajes de bajo nivel y la percepción de que los métodos formales obstaculizan la inves-

tigación emergen como factores críticos [3]. Esto llevó a la formulación de un proyecto de investigación (aún en desarrollo) con docentes investigadores en astrofísica y ciencias de la computación de la Insitución Universitaria Politécnico Grancolombiano, la Institución Universitaria Pascual Bravo y la Universidad de Antioquia para la construcción de un material rápido, efectivo y de calidad sufiente para que un científico en formación o ejercicio pueda mejorar sus prácticas de desarrollo de software desde una perspectiva de la Ingenieria del Software. Su propósito es ofrecer un camino de adopción progresiva de buenas prácticas —sin restringir el trabajo investigativo— que permita aumentar la calidad, la trazabilidad y la reproducibilidad del software empleado en investigación.

La guía se apoya en siete principios clave de la ingeniería de software: (i) **Modularidad** (separación de responsabilidades y contratos de interfaz), (ii) **Mantenibilidad** (manejo eficiente de comprensión, corrección y actualizaciones), (iii) **Escalibidad** (entornos y dependencias declaradas con opción de crecer en tamaño y complejidad), (iv) **Eficiencia** (optimización de los recursos utilizados en el software), (v) **Usabilidad** (diseño intuitivo de la experiencia de usuario), (vi) **Confiabilidad** (pruebas unitarias, de regresión y criterios de aceptación) y (vii) **Portabilidad** (adaptación a diferentes entornos).

Operativamente, la guía organiza el trabajo en fases con *artefactos mínimos* por cada una:

1. **Planificación**: Especificación de Requisitos de Software (SRS) breve, con propósito, alcance, historias de usuario y criterios de aceptación verificables.

2. **Diseño**: Diagramas de contexto, componentes y clases/flujo de datos; definición de formatos de entrada/salida y contratos de módulos.

3. **Implementación**: Estructura de paquete/carpetas, convenciones de estilo, gestión explícita de dependencias y parámetros.

4. **Pruebas**: Pruebas unitarias mínimas por módulo, pruebas de regresión numérica donde aplique y mapeo de cada criterio de aceptación a evidencia de prueba.

5. **Mantenimiento**: guiones de *benchmarking* y perfilamiento, y checklist de mantenibilidad.

En términos conceptuales, la guía operacionaliza los criterios de **eficiencia** y **calidad** discutidos en este marco: (a) reduce ambigüedad y *retrabajo* con el front-loading de requisitos y diseño; (b) mejora **mantenibilidad** mediante modularidad y documentación viva; (c) favorece la **reproducibilidad** al exigir entornos declarados y rutas de ejecución claras; y (d) habilita **verificación/validación** continua con criterios de aceptación mapeados a pruebas. Así, "estándares de calidad" se entienden aquí como un conjunto mínimo verificable de prácticas y artefactos que elevan corrección, confiabilidad, claridad, reproducibilidad y reutilización del software científico.

Estado de desarrollo e iteración: La guía no es una versión definitiva: se encuentra en proceso de refinamiento continuo. Los resultados de este trabajo de grado se incorporan como *evidencia empírica* para iterar su contenido (plantillas, checklists, rúbricas y ejemplos por dominio), con el objetivo de converger hacia una versión estable que aporte conceptos y herramientas concretas para que científicos desarrollen con estándares de calidad. En consecuencia, los hallazgos aquí presentados deben leerse como parte de un ciclo iterativo de diseño–evaluación–mejora, más que como una validación final.

Alcance y limitaciones: La guía es *un marco de procesos* aplicable a cualquier campo científico y no prescribe técnicas numéricas específicas; provee un andamiaje de proceso y artefactos mínimos. No sustituye el juicio científico ni la revisión por pares del contenido disciplinar; su foco es estructurar el *cómo* se construye y verifica el software, dejando a cada proyecto la adaptación necesaria según el problema físico y sus datos.

IV. OBJETIVOS

A. Objetivo general

- Evaluar el efecto de una guía metodológica para el desarrollo de software científico sobre la eficiencia del proceso y la calidad del software realizado por estudiantes de Astronomía.

B. Objetivos específicos

- **Examinar la guía:** revisar su alcance, coherencia y alineación con buenas prácticas; identificar vacíos y ajustes necesarios.
- **Diseñar el experimento computacional a realizar:** definir el problema, las condiciones/datos de entrada, el entorno de ejecución y criterios claros de evaluación.
- **Diseñar el experimento para medir el impacto de la guía:** establecer grupos (control/experimental), instrumentos de medición y un plan básico de análisis.
- **Medir la contribución de la guía al proceso de desarrollo de software científico:** recolectar métricas (tiempo, calidad, rendimiento) y *triangular* la evidencia cuantitativa y cualitativa para identificar oportunidades de mejora de la guía y de la enseñanza de desarrollo de software científico.

V. PROBLEMA DE INVESTIGACIÓN

Aunque se reconoce que las buenas prácticas incrementan la reproducibilidad y la mantenibilidad, no está claro si su incorporación formal durante la etapa de formación universitaria conlleva una mejora significativa en la calidad de los productos y en la eficiencia del proceso. Según Barker[11], la producción científica contemporánea depende crecientemente del software de investigación, y diversos marcos y recomendaciones—como los principios FAIR4RS—promueven prácticas que favorecen la reproducibilidad, la mantenibilidad y la reutilización del código. Aun así, en el ámbito académico persisten brechas de formación y adopción sistemática de ingeniería de software, lo que impacta la calidad y sostenibilidad del software utilizado en investigación [7, 21]. Estudios recientes caracterizan problemas técnicos que obstaculizan la reproducibilidad (p.,ej., *reproducibility debt*) y subrayan la relación entre reproducibilidad y mantenimiento sostenido del software [15, 22].

Aunque existe consenso sobre los beneficios esperados de las buenas prácticas, la evidencia empírica comparativa—con diseños controlados—que mida el efecto causal de incorporar formalmente un marco metodológico de desarrollo de software científico durante la formación universitaria sigue siendo limitada. La literatura propone integrar explícitamente entrenamiento en reproducibilidad y desarrollo de software en los planes de estudio [23], y empiezan a aparecer estudios de intervención en cursos que muestran mejoras en mantenibilidad y confiabilidad al introducir prácticas y herramientas de análisis estático; sin embargo, su alcance permanece acotado y no se centra específicamente en marcos integrales para software científico [24]. De esta manera surge la siguiente pregunta de investigación:

Pregunta de investigación

¿En qué medida la implementación de una guía metodológica de desarrollo de software científico influye en la eficiencia del proceso y en la calidad del software elaborado por estudiantes universitarios?

La respuesta a esta pregunta permitirá fundamentar la inclusión de marcos formales en los planes de estudios y ofrecerá directrices para optimizar la enseñanza de programación científica.

VI. METODOLOGÍA

A. *Estudio preliminar de casos individuales guiados.*

1) Propósito y alcance: Para dotar de fundamento empírico y afinar el diseño del experimento comparativo, se incorporan dos **estudios de caso individuales guiados** que operan como *pilotos metodológicos* y fuente de evidencia cualitativa estructurada. El primero corresponde al desarrollo, por parte del autor de este documento, de un *software que compara métodos de integración numérica*; el segundo, al desarrollo de un *simulador de fluidos mediante la técnica SPH (Smoothed Particle Hydrodynamics)* realizado por un estudiante de último semestre del pregrado de Astronomía de la Universidad de Antioquia. Estos casos persiguen: (i) *operacionalizar* la *Guía Instructiva y Metodológica para el Desarrollo de Software Científico* en contextos reales, (ii) documentar—mediante artefactos y reflexión sistemática—su efecto sobre el *proceso* (planificación, diseño, implementación, verificación) y el *producto* (modularidad, documentación, reproducibilidad), e (iii) identificar beneficios y costos percibidos, así como cuellos de botella, en dimensiones clave (*eficiencia*, *calidad estructural*, *reproducibilidad* y *mantenibilidad*). Las evidencias consideradas incluyen SRS, diagramas de diseño, estructura de paquetes, pruebas y bitácoras reflexivas; sus hallazgos alimentarán la iteración de la guía y la calibración de los instrumentos del estudio (rúbrica, registro de horas y encuesta) -definidos más adelante-, aportando insumos directos para la definición de criterios de aceptación y su trazabilidad a pruebas.

2) Diseño y postura metodológica: Se abordan dos **estudios de caso instrumentales** integrados a un *diseño mixto*. La evidencia cualitativa obtenida en cada caso complementa y explica los hallazgos cuantitativos del trabajo. Se adopta una **postura reflexiva** del autor (en el Caso A, rol de investigador–desarrollador) y una **observación guiada** en el Caso B (acompañamiento del investigador al estudiante). Para mitigar sesgos, se emplean: (i) trazabilidad explícita de decisiones, (ii) triangulación de fuentes (artefactos, código, pruebas y bitácora) y (iii) registro sistemático por iteraciones.

3) Preguntas orientadoras:

- ¿Cómo incide la guía en la **reducción de ambigüedad** inicial y la **trazabilidad** requisito→código→pruebas?
- ¿Qué efectos tiene sobre la **modularidad** y la **extensibilidad** de la solución?
- ¿En qué medida promueve **verificación temprana** y **reproducibilidad** operativa (entorno, dependencias, datos)?
- ¿Qué **costos percibidos** introduce (p.,ej., documentación/diagramación) y cómo se compensan durante el ciclo?

Foco por caso. **Caso A (Software de comparación de integradores numéricos):** extensibilidad de *integradores* y contrato de interfaces. **Caso B (simulador SPH):** organización por *dominios*, kernels/vecindad, condiciones iniciales y visualización.

4) Unidades de análisis y evidencias:

- **Evidencia documental**: SRS (propósito, alcance, historias, criterios de aceptación), diagramas (contexto, componentes, clases/flujo, formatos).
- **Evidencia de implementación**: estructura de paquete y módulos.
  - *Caso A*: módulos de integradores con interfaz común; utilidades de E/S y visualización; orquestación.
  - *Caso B*: módulos SPH (kernels, vecindad), dominios y *boundaries*; generadores de *ICs* y visualizador.
- **Evidencia operativa y perceptual**: bitácora por iteraciones (decisiones, alternativas, criterios), tiempos por fase y autoevaluación guiada por rúbrica.

5) Instrumentos:

- **Bitácora del desarrollo** (por iteración): fecha, fase, decisión, alternativas, criterio, tiempo (h).
- **Matriz de trazabilidad**: Requisito → Módulo/Clase → Prueba/Evidencia.
- **Checklist de reproducibilidad mínima**: archivo de dependencias, instrucciones de ejecución, formato de datos, *seed* y parámetros por defecto.

6) Rúbrica preliminar de evaluación técnica: Con base en estos instrumentos se planteó el diseño de una rúbrica de evaluación técnica que permitirá medir aspectos como:

- Organización del código (modularidad, limpieza)[25, 24, 10]
- Documentación generada (claridad, estructura, uso) [11, 10]
- Funcionalidad del software (cumplimiento de requisitos)[25, 26]
- Rendimiento (tiempo de simulación, eficiencia computacional)[25, 27, 28]
- Facilidad de mantenimiento (modificabilidad del código)[25, 24, 10]

a) Escala, cómputo e interpretación: La calificación global se expresa en una **escala de 0 a 100 puntos** obtenida como suma de cinco dimensiones con tope por criterio: *Organización del código* (0–15), *Documentación* (0–15), *Funcionalidad* (0–40), *Rendimiento* (0–20) y *Mantenibilidad* (0–10). Formalmente,

$$\text{Puntaje total} = O_{(0\text{–}15)} + D_{(0\text{–}15)} + F_{(0\text{–}40)} + R_{(0\text{–}20)} + M_{(0\text{–}10)} \in [0, 100].$$

0 puntos indica *ausencia de evidencia suficiente* y/o *incumplimiento crítico* en todas las dimensiones (p.,ej., código desorganizado, sin documentación, sin verificación ni reproducibilidad). **100 puntos** corresponde a *cumplimiento pleno y consistente* en todas las dimensiones, *con evidencias verificables* (SRS, diagramas, pruebas, README ejecutable, entorno declarado) y calidad técnica demostrada (modularidad, trazabilidad, desempeño medido, facilidad de cambio).

b) Bandas de desempeño (guía de lectura):

- **0–19 (Deficiente):** evidencias nulas o insuficientes; fallas sustantivas.
- **20–39 (Inicial):** cobertura mínima y fragmentaria; alto riesgo de retrabajo.
- **40–59 (Aceptable):** criterios básicos presentes con lagunas; reproducibilidad parcial.
- **60–79 (Bueno):** cumplimiento mayoritario y consistente; reproducibilidad y verificación adecuadas.
- **80–100 (Excelente):** cumplimiento integral con buenas prácticas sostenidas; documentación viva, pruebas y mediciones de rendimiento alineadas a criterios de aceptación.

La asignación dentro de cada dimensión se ancla en: (i) *grado de cumplimiento* de los criterios de aceptación, (ii) *calidad técnica* observable (p.,ej., modularidad, complejidad razonable, contratos de interfaz), y (iii) *evidencia verificable* de reproducibilidad y pruebas.

7) Procedimiento:

- **Requisitos**: consolidación de SRS (propósito, alcance, restricciones, historias y criterios).
- **Diseño**: diagramas esenciales (contexto, componentes, clases/flujo, formatos) y definición de contratos de módulo.
- **Implementación incremental**:
  - *Caso A*: incorporación de integradores con interfaz común y utilidades; orquestación del flujo.
  - *Caso B*: construcción de kernels/vecindad y *ICs*; separación de dominios y *boundaries*.
- **Verificación temprana**:
  - *Caso A*: pruebas mínimas por integrador (avance de estado y propiedades básicas).
  - *Caso B*: casos de referencia acotados (p.,ej., configuraciones simples y chequeos de consistencia local) y validaciones de salida del generador de *ICs*.
- **Documentación y empaquetado**: README ejecutable, dependencias, datos/formatos y parámetros por defecto; breve manual de uso.
- **Registro reflexivo**: cierre de cada iteración con actualización de bitácora y autoevaluación breve en rúbrica.

8) Análisis y triangulación: Se aplica **codificación temática ligera** a la bitácora (ambigüedad/trazabilidad; modularidad/extensibilidad; verificación; reproducibilidad; costos/beneficios) y se vincula cada *insight* con evidencia (SRS, diagramas, código, pruebas). La **triangulación** contrasta: (i) artefactos documentales, (ii) ejecución de pruebas y (iii) métricas operativas registradas.

9) Salida esperada: Para cada caso se entrega una **síntesis técnica** (artefactos clave, estructura del paquete, pruebas y reproducibilidad), **efectos observados** por dimensión, una **evaluación aplicando la rúbrica** para calibrar los posibles resultados de las siguientes iteraciones, **limitaciones** y **oportunidades de mejora**. Se incluye una **conclusión reflexiva** que consolida lecciones aprendidas y recomendaciones para el diseño del estudio comparativo y la iteración de la guía.

B. ***Proyecto de desarrollo de software: Análisis de estabilidad en galaxias de disco simuladas.***

La presente investigación adopta un enfoque cuantitativo-comparativo, con elementos cualitativos, orientado a verificar la eficiencia de una guía metodológica para el desarrollo de software científico en un contexto educativo. Para ello, se ha diseñado un estudio aplicado con estudiantes universitarios, divididos en dos grupos, quienes desarrollarán un mismo proyecto de software bajo condiciones controladas pero diferenciadas en cuanto al uso de una guía estructurada de desarrollo.

1) Diseño del estudio: Se emplea un diseño cuasi-experimental de grupos no aleatorizados, en el cual se comparan dos grupos de estudiantes:

- Grupo A (experimental): desarrolla el proyecto con apoyo de una guía metodológica de desarrollo de software científico.
- Grupo B (control): desarrolla el mismo proyecto sin acceso a dicha guía, utilizando su propio criterio y conocimientos previos.

Este diseño permite observar el impacto de la intervención (la guía) sobre la eficiencia del desarrollo y la calidad del producto final, bajo condiciones similares de tiempo, herramientas y contenidos.

2) Participantes: La muestra está compuesta por estudiantes de nivel universitario del pregrado de Astronomía de la Universidad de Antioquia con un rango de edad que oscila entre los 22 y 29 años; con conocimientos básicos en programación, estructuras de datos y conceptos de física computacional. Se conformaron dos grupos de trabajo equilibrados en cuanto a tamaño y nivel académico, repartidos entre tercer y séptimo nivel del pregrado, con el fin de reducir sesgos en la comparación. La participación de los estudiante fue voluntaria y la realización de este proyecto está por fuera de su plan de estudios, por lo que no existieron intereses relacionados compromisos académicos de por medio. Estos han firmado un acuerdo de confidencialidad con el fin de proteger su identidad.

3) Proyecto de desarrollo: El proyecto que se asignará a ambos grupos consiste en diseñar e implementar un aplicativo de escritorio para el análisis de la estabilidad en galaxias de disco simuladas. La aplicación cubrirá todo el flujo de trabajo —desde el ingreso de datos hasta la exportación de resultados— y reproducirá los retos habituales del desarrollo de software científico.

1. **Interfaz gráfica de usuario (GUI).** Debe ofrecer controles intuitivos para cada etapa del análisis sin recurrir a la línea de comandos.
2. **Carga flexible de datos en formato *Gadget*.** El usuario podrá abrir archivos ASCII o binarios en las versiones 1 y 2 del formato Gadget, los cuales se pre-procesarán automáticamente.
3. **Pre-procesamiento geométrico.** El sistema rotará las partículas para alinear el eje del disco galáctivo con el eje Z, asegurando una orientación física adecuada que facilita la interpretación de los resultados finales.
4. **Selección de componentes y vistas ortogonales.** El usuario podrá analizar por separado disco, bulbo o halo y visualizar sus proyecciones XY, XZ, YZ.
5. **Módulo de análisis configurable.** Desde la GUI se elegirán los análisis (p. ej. perfiles de masa, cinemática, potencial), ajustando secuencialmente sus parámetros y mostrando los resultados con sus respectivas curvas de ajuste y estimativos de error.
6. **Algoritmos científicos obligatorios.**
   - Perfil vertical de masa del disco. Modelos isoterma/sech^2 y revisiones observacionales/teóricas [29, 30, 31].
   - Perfil superficial de masa del disco. Disco exponencial y variantes, formulaciones estándar [30, 31].
   - Perfil volumétrico de masa de componentes esféricas. Bulbo/halo con perfiles analíticos típicos (Hernquist, NFW) [32, 33, 31].
   - Dispersión de velocidades σ_R, σ_ϕ, σ_Z. A partir de ecuaciones de Jeans (y esquemas JAM) [31, 34].
   - Potencial gravitacional radial. Relación $v_c^2 = R\,\partial\Phi/\partial R$ y derivaciones [31].

- Frecuencia de epiciclo $\kappa(R)$. Definición $\kappa^2 = R\,\mathrm{d}\Omega^2/\mathrm{d}R + 4\,\Omega^2$ [31].
- Parámetro de estabilidad de Toomre $Q(R)$. $Q = \sigma_R\kappa/(3{,}36\,G\,\Sigma)$, formulación original y tratamientos modernos [35, 31].

7. **Configuración global y gestión de parámetros.** Antes de cada lote de análisis se definirán umbrales globales (radio máx., `z` máx., método de *binning*) para asegurar consistencia.
8. **Exportación de resultados.** El simulador permitirá descargar tablas y gráficos.
9. **Transparencia y atribución.** Cada salida incluirá la descripción del algoritmo utilizado y la información del desarrollador para preservar trazabilidad científica.

4) Entregables y criterios de evaluación:

- **Código reproducible** en Python con control de versiones, pruebas unitarias y documentación. Este lenguaje de programación se elige ya que es el más cercano a los estudiantes en su proceso formativo, además, por su versatilidad para afrontar problemas computacionales científicos.
- **Informe técnico** con la arquitectura, decisiones de diseño y métricas de rendimiento (tiempo de desarrollo, mantenibilidad, uso de memoria).

Los entregables permitirán evaluar la eficiencia de manera *cuantitativa* (cobertura de pruebas, complejidad ciclomática) y *cualitativa* (experiencia del desarrollador-investigador), alineando así el proyecto con los objetivos de verificación de la guía metodológica.

5) Instrumentos de medición: Para evaluar el desempeño de cada grupo y verificar la eficiencia de la guía aplicada, se utilizarán los siguientes instrumentos:

a) Registro de tiempo de desarrollo: Se recopilan datos del tiempo invertido por cada equipo, segmentado por tipo de actividad:

- **Apropiación teórica**: revisión de fundamentos físicos/matemáticos y de la literatura relevante del problema.
- **Lectura y comprensión de la guía**: estudio de la guía metodológica y adaptación de sus lineamientos al proyecto.

- **Planificación**: definición de objetivos de iteración, descomposición de tareas, estimación y criterios de aceptación.
- **Diseño**: elaboración de SRS/artefactos de diseño (diagramas, interfaces, formatos de datos) y decisiones arquitectónicas.
- **Implementación**: construcción de módulos, scripts, integración y/o GUI.
- **Pruebas**: diseño y ejecución de pruebas (unitarias, de integración o regresión) y verificación de criterios de aceptación.
- **Mantenimiento**: refactorización, corrección de defectos y pequeños ajustes derivados de retroalimentación.

Cada uno de los diferentes tipos de actividades tiene procesos asociados a cada uno (Descripción, herramientas utilizadas, Dificultades encontradas y observaciones).

b) Rúbrica de evaluación técnica: Con la construcción de la *rúbrica piloto* en los estudios de caso (software de comparación de métodos de integración numérica y simulador SPH), se plantea **aplicarla de forma estandarizada** en este proyecto comparativo para ambos grupos (con y sin guía). La rúbrica valora cinco dimensiones —*organización del código, documentación, funcionalidad, rendimiento y mantenibilidad*— mediante criterios observables y evidencias verificables (p.,ej., SRS, diagramas, estructura de paquetes, pruebas y README ejecutable). Para cada entregable se registrará el **puntaje por criterio** y una **justificación breve** anclada en evidencias; los resultados se utilizarán en el *análisis comparativo* y en la *triangulación* con el registro de horas y la encuesta de percepción.

c) Encuesta de percepción post-proyecto: Con el objetivo de recoger impresiones sobre el proceso y, en el grupo experimental, sobre la guía utilizada, se aplicará un cuestionario al finalizar el proyecto. La participación en el mismo es *anónima*, *voluntaria* y de *carácter formativo*. La estructura incluye: (i) perfil y experiencia previa; (ii) adopción de prácticas durante el desarrollo (escala 1–5: Nunca–Siempre) —documentación, pruebas, planificación, control de versiones—; (iii) percepción de la guía (escala 1–5 en claridad, aplicabilidad, apoyo a la mantenibilidad y verificación) y estimación de impacto global (escala 0–10); (iv) valoración general de la experiencia (reproducibilidad, organización por fases, confianza en el estado del código); y (v) preguntas abiertas para mejoras.

6) Procedimiento:

- Introducción del proyecto y asignación de grupos. en esta parte todos los participantes estuvieron en un proceso formativo y de nivelación en competencias y herramientas computacionales en Python, así, los dos grupos tienen el mismo punto de partida a nivel formativo.
- Socialización de la guía al grupo A (entregada al inicio del proyecto de desarrollo).
- Desarrollo del proyecto en paralelo durante el mismo periodo de tiempo.
- Recolección de productos finales y aplicación de los instrumentos de medición.
- Análisis comparativo de los resultados entre los dos grupos.

7) Análisis de resultados: Los datos cuantitativos se analizarán mediante estadística descriptiva y comparativa. Para ello, se analizan los principales estadísticos derivados de las horas reportadas por cada estudiante, lo que facilita comparar el esfuerzo y su variabilidad entre cohortes.

- *Grupo*: cohorte analizada (Control / Experimental).
- $\boldsymbol{n}$: número de participantes en cada grupo.
- *Suma [h]*: horas totales registradas por el grupo.
- *Media [h]*: promedio de horas por estudiante del grupo.
- *Mediana [h]*: valor central de horas al ordenar a los estudiantes del grupo.
- *Desv. [h]*: desviación estándar de las horas entre estudiantes del grupo.
- *Mín. [h]*: menor número de horas registradas por un estudiante del grupo.
- *Máx. [h]*: mayor número de horas registradas por un estudiante del grupo.

Adicionalmente, se plantea analizar la distribución del tiempo en porcentaje por tipo de actividad, con el fin de caracterizar cómo se reparte el esfuerzo a lo largo del ciclo de vida del proyecto. Las categorías consideradas fueron:

- **Apropiación teórica / Lectura y comprensión de la guía / Planificación/ Diseño / Implementación/ Pruebas / Mantenimiento**

Los datos cualitativos obtenidos a partir de la encuesta se emplearán para *triangular* la evidencia cuantitativa (registro de horas y rúbrica técnica) e identificar oportunidades de mejora de la guía y de la enseñanza de desarrollo de software científico.

VII. RESULTADOS

A. ***Estudio de caso personal: aplicación de la guía en un software de comparación métodos de integración numérica.***

Con el fin de complementar los resultados cuantitativos del estudio comparativo, el autor de este documento desarrolló un *software que compara diferentes métodos de integración numérica* siguiendo la *Guía Instructiva y Metodológica para el Desarrollo de Software Científico*. Este ejercicio se reporta aquí como un resultado cualitativo y reflexivo sobre la eficacia práctica de la guía en un caso real.

1) Artefactos producidos: El estudio de caso generó tres conjuntos de artefactos:

- **Documentación y requisitos**: Especificación de Requisitos de Software (SRS) con propósito, alcance, restricciones, historias de usuario, casos de uso y criterios de aceptación; define explícitamente los cuatro integradores a implementar (Euler, Leapfrog, RK8 y Bulirsch–Stoer) y límites de desempeño esperados.
- **Diseño**: Descomposición funcional, diagrama de contexto, diagrama de componentes, diagrama de clases (UML) y diagramas de secuencia, además de una nota de formatos de datos.
- **Implementación** : Paquete Python modular, con subpaquetes, archivo para facilitar la reproducibilidad de tipo `requirements.txt`, y un `README.md` con instrucciones de uso.

2) Síntesis técnica de la implementación: La estructura del paquete refleja la separación de responsabilidades promovida por la guía:

- **Integradores:** 4 archivos `.py` cada uno con los integradores de manera independiente que heredan/usan una base común para homogenizar la interfaz de `step()`.
- **Modelos:** `particula.py` y `sistema.py` encapsulan el estado (`SystemState`) y entidades (`Particula`).
- **Utilidades:** 2 archivos `.py`, uno para la carga de archivos en formato `.csv` y otro para la visualización 2D de las trayectorias.

- **Orquestación:** `main.py` permite seleccionar el método e integrar un flujo de carga → simulación → visualización.
- **Pruebas:** Una prueba parametrizada con `pytest` que verifica, para los cuatro integradores, que una partícula avanza de estado en un paso fijo y preserva el número de partículas.

3) Cómo la guía influyó en el proceso y el producto: Siguiendo las fases propuestas (requisitos → diseño → implementación → verificación → documentación), los principales efectos observados fueron:

- **Trazabilidad de requisitos a código:** las historias de usuario del SRS se mapearon explícitamente a módulos (p.ej., *"comparar métodos"* → selección en `main.py` y visualización superpuesta en `visualizacion.py`). Esta trazabilidad facilitó priorizar funciones mínimas y evitar derivas de alcance.
- **Modularidad y extensibilidad:** el diseño con `integradores/` permitió añadir y probar variantes sin romper el resto del sistema (sustitución de integradores por interfaz común). Esto redujo el acoplamiento y favorece la *experimentación controlada* esperada en software científico.
- **Verificación temprana:** aunque las pruebas son incipientes, el caso de `pytest` detectó regresiones al refactorizar `SystemState`. La guía incentivó incluir la carpeta `tests/` desde el inicio.
- **Reproducibilidad operativa:** la presencia de `requirements.txt` y un `README.md` ejecutable mejora la repetibilidad del flujo (instalación, formato de entrada, ejecución). En trabajos estudiantiles esto suele faltar, y la guía lo hizo explícito.
- **Calidad del diseño:** los diagramas (contexto, componentes, clases y secuencia) aceleraron el alineamiento entre modelo conceptual y estructura del paquete; disminuir la ambigüedad de interfaces ayudó a evitar retrabajo.

4) Limitaciones y oportunidades de mejora detectadas: A pesar de los avances, identifiqué aspectos por fortalecer en iteraciones futuras (también guiados por la propia guía):

- **Documentación interna:** baja densidad de docstrings a nivel de módulos/clases/funciones. Incorporar docstrings con ejemplos (`doctest`) y tipado estático (`typing`) mejoraría la

mantenibilidad.

- **Cobertura de pruebas:** extender `pytest` con aserciones físicas (conservación de energía/momento para Leapfrog) y pruebas de regresión numérica permitiría validar propiedades de los integradores más allá de "avanza/ no avanza".
- **Separación de parámetros físicos:** centralizar constantes (p.ej., G, M) en un módulo de configuración para facilitar barridos y reproducibilidad de escenarios.
- **Rendimiento y escalabilidad:** medir tiempos y uso de memoria en escenarios sintéticos (100–1000 partículas, 10^3–10^4 pasos) y perfilar cuellos de botella (vectorización, `numba`) como lo recomienda la guía para la fase de optimización.

5) Resultados de la rúbrica aplicada: Siguiendo los criterios de evaluación propuestos en la rúbrica, la Tabla I muestra los resultados del software de comparación de métodos numéricos desarrollado por el autor del texto.

TABLA I
RESULTADOS DE EVALUACIÓN POR CRITERIOS — SOFTWARE COMPARATIVO DE MÉTODOS NUMÉRICOS

Criterio	**Puntaje**	**Fundamento (evidencia)**
Organización del código (0–15)	**13**	Paquete con separación de responsabilidades: `integradores/` (Euler, Leapfrog, RK8, Bulirsch–Stoer con `base.py`), `modelos/` (estado y entidades), `utils/` (E/S y visualización) y `tests/`. 11 archivos `.py`, ∼279 LOC; estructura clara y extensible.
Documentación generada (0–15)	**12**	`README.md` y `requirements.txt` presentes (reproducibilidad operativa); densidad de `docstrings` baja a nivel de módulos/clases/funciones; faltan ejemplos embebidos/guía de uso ampliada.
Funcionalidad (0–40)	**38**	Selección y ejecución de 4 integradores con visualización de trayectorias; flujo carga → simulación → visualización; prueba básica por integrador (`pytest`) verifica avance de estado. Faltan casos de referencia más ricos y validaciones físicas de mayor alcance.
Rendimiento (0–20)	**16**	Implementación vectorizada moderada; sin `numba`/JIT ni *benchmarks* o perfilamiento sistemático (tiempo por paso vs. N, uso de memoria).
Mantenibilidad (0–10)	**7**	Interfaz común de integradores favorece extensibilidad; presencia de `tests/` mínimos. Oportunidades: aumentar cobertura de pruebas, tipado estático (`typing`) y `docstrings` con ejemplos.
Total	**86/100**	

6) Apreciación personal del autor respecto al proyecto: La experiencia directa aplicando la guía fue positiva: *disminuyó la ambigüedad* al comienzo, *estructuró* la incorporación ordenada de métodos numéricos e *incitó* a integrar verificación y empaquetado mínimo desde el inicio. El costo adicional inicial (diagramar y escribir el SRS) se compensó con menos retrabajo y mayor claridad de interfaces. En términos de los criterios del estudio, se estiman mejoras en **eficiencia del proceso** (menos iteraciones ad hoc), **calidad estructural** (paquete modular con responsabilidades claras) y **reproducibilidad** (entorno declarativo), quedando como tarea aumentar la **cobertura de pruebas** y la **documentación interna**.

B. ***Estudio de caso externo: uso de la guía en el proyecto SPH***

Adicionalmente, con el objetivo de complementar los resultados del presente estudio comparativo. Se evaluó la aplicación de la guía en el desarrollo de un **proyecto de un simulador de fluidos usando la técnica de SPH (*Smoothed Particle Hydrodynamics*)**, realizado por un estudiante de último semestre del pregrado de Astronomía de la Universidad de Antioquia, cuyos artefactos se organizaron en dos frentes: Documentación del diseño del proyecto e Implementación. La evaluación se efectuó con la **rúbrica técnica** del estudio.

1) Cobertura funcional observada: La estructura del proyecto se centra en la **generación y manejo de condiciones iniciales** y en utilidades asociadas: módulos de *kernels* y vecindad, geometría del fluido, dominios y *boundaries*, construcción de configuraciones de partículas y su visualización.
No se identificaron: (i) **pruebas automatizadas** (*harnesses*) y (ii) **módulos de integración temporal** que evolucionen el fluido paso a paso preservando invariantes físicos (p. ej., masa, energía o momento). En consecuencia, el alcance actual cubre principalmente el **estado inicial** del sistema, más que su **evolución dinámica** en el tiempo.

2) Resultados de la rúbrica aplicada: Con el objetivo de cuantificar la aplicación y efecto de la guía en el presente caso de estudio, la Tabla II muestra los resultados de la rúbrica aplicada al proyecto del simulador SPH.

TABLA II
RESULTADOS DE EVALUACIÓN POR CRITERIOS - SIMULADOR SPH.

Criterio	**Puntaje**	**Fundamento (evidencia)**
Organización del código (0–15)	**14**	Paquetería con `__init__.py`, submódulos por dominio (`domains`, `fluid`, `boundaries`); separación de responsabilidades; ausencia de `tests/` formales.
Documentación generada (0–15)	**10**	Diseño documentado; `README.md` parcial en `initial_conditions`; falta `requirements.txt/pyproject.toml` y baja densidad de `docstrings`.
Funcionalidad (0–40)	**32**	Módulos para *ICs*, kernels y utilidades de SPH; sin *testbed* formal ni casos de referencia (p. ej. colisión 1D).
Rendimiento (0–20)	**13**	Vectorización vía `NumPy`; no se detecta `Numba` ni perfilamiento/benchmarks; sin medición de escalabilidad (N, pasos).
Mantenibilidad (0–10)	**7**	Estructura modular y nombres consistentes; sin `tests` ni archivo de entorno; tipado estático limitado.
Total	**76/100**	

3) Apreciación personal del proyecto: La guía *ordenó* el trabajo desde el diseño (diagramas y descomposición por dominios) y facilitó construir un paquete reutilizable de *condiciones iniciales* con visualizador asociado. El costo inicial en documentación y organización se compensó al reutilizar *builders* y *exporters* entre escenarios. Persisten mejoras clave: consolidar un **grupo de pruebas automatizadas** (invariantes físicos y regresión numérica), definir un **entorno reproducible** (`requirements.txt/environment.yml`) y establecer **benchmarks mínimos** (tiempo por paso vs. N, memoria).

C. ***Proyecto de Análisis de estabilidad de galaxias de disco simuladas.***

En esta sección se presentan los resultados **agregados por grupo**—*Control* (sin guía) y *Experimental* (con guía)—y, de forma específica, la **comparación del par** que desarrolló el mismo proyecto bajo condiciones distintas (con guía vs. sin guía).

1) Descripción de la muestra: Se analizaron **2** integrantes del **grupo Control** y **2** del **grupo Experimental**. *Inicialmente* se planearon **4** participantes por grupo; sin embargo, durante la ejecución del estudio se presentó **deserción** por priorización de otras actividades académicas y/o personales, lo que redujo la muestra efectiva a **2+2**. Este recorte refleja, en parte, la débil cultura de *sostenimiento de prácticas y dedicación continuada* en proyectos de desarrollo de software con estándares de calidad que este trabajo precisamente busca fomentar. Dado el tamaño muestral y la asimetría observada en las distribuciones, se reportan medidas robustas (medianas) y sensibles a valores extremos (medias), además de dispersión.

2) Esfuerzo total por grupo en términos de horas invertidas en el proyecto: La Tabla III resume horas totales registradas por participante y los descriptivos sobre dichas horas.
Lecturas principales.

- La **mediana** del *Experimental* (44,38 h) es menor que la del *Control* (57,28 h), sugiriendo—de forma *preliminar*—**menor tiempo típico** por estudiante cuando se usa la guía.
- La **media** del *Experimental* (44,38 h) también es menor que la del *Control* (57,28 h); la

TABLA III

HORAS POR GRUPO (DESCRIPTIVOS SOBRE HORAS TOTALES POR ESTUDIANTE).

Grupo	n	**Suma [h]**	**Media [h]**	**Mediana [h]**	**Desv. [h]**	**Mín. [h]**	**Máx. [h]**
Control	2	114.57	57.28	57.28	19.25	43.67	70.90
Experimental	2	88.75	44.38	44.38	3.71	41.75	47.00

dispersión en el Experimental es baja ($SD = 3{,}71$ h) frente al Control ($SD = 19{,}25$ h).

- Diferencia de medias (Control–Experimental): +12,91 h (+29,1 %). El delta de Cliff (Control vs. Experimental) es $\delta \approx 0{,}50$ (*mediano*), consistente con que los valores del Control tienden a superar a los del Experimental.

3) Distribución del tiempo por tipo de actividad: La Tabla IV muestra la **proporción** de horas por categoría

TABLA IV

DISTRIBUCIÓN DE TIEMPO POR TIPO DE ACTIVIDAD (PORCENTAJE).

Control (sin guía)

Actividad	%
Diseño	14.1 %
Implementación	58.8 %
Lectura/Estudio	16.5 %
Mantenimiento	2.1 %
Planificación	6.5 %
Pruebas	2.0 %

Experimental (con guía)

Actividad	%
Diseño	57.1 %
Implementación	22.9 %
Lectura/Estudio	14.4 %
Mantenimiento	0.0 %
Planificación	5.6 %
Pruebas	0.0 %

Lecturas principales.

- El **Experimental** concentra el esfuerzo en **Diseño** (57.1 %) y **Planificación** (5.6 %); en conjunto, **fases tempranas** suman **62.7 %**.
- El **Control** destina la mayor fracción a **Implementación** (58.8 %) y solo **20.6 %** a fases tempranas (Diseño+Planificación).
- La proporción en **Pruebas** es baja en ambos grupos; es un foco de mejora inmediato.

4) Rúbrica técnica comparativa: La Tabla V sintetiza y compara las evaluaciones de la rúbrica aplicada a cada grupo - experimental (con guía) y de control (sin guía)- respectivamente:

TABLA V
RÚBRICA TÉCNICA COMPARATIVA.

Criterio	**Con guía**	**Sin guía**
Organización del código (0–15)	**12**	**6**
Documentación generada (0–15)	**15**	**5**
Funcionalidad (0–40)	**38**	**32**
Rendimiento (0–20)	**18**	**11**
Mantenibilidad (0–10)	**6**	**4**
Total (0–100)	**89**	**58**

Interpretación focalizada de la comparativa:

- **Con guía** obtiene ventaja marcada en **documentación y trazabilidad** (15/15), **funcionalidad** (38/40) y **organización** (12/15), consistente con mayor inversión en fases tempranas.
- **Sin guía** concentra esfuerzo en implementación, con **docstring nulo** o bajo, sin empaquetado reproducible ni `tests`, lo que penaliza **mantenibilidad** y **reproducibilidad** (reflejado en documentación y organización).
- En **rendimiento**, ninguno instrumenta *benchmarks* formales; sin embargo, el caso con guía deja trazado para incorporar mediciones y optimización. Si bien ninguno de los proyectos se ha entregado en su totalidad y las calificaciones de la rúbrica corresponden a evaluaciones parciales por limitaciones de tiempo, se evidencia mayor eficiencia en las implementaciones esenciales por parte del grupo con guía.

D. Percepción de los participantes (encuesta post-proyecto)

Con el fin de triangular los hallazgos cuantitativos con evidencia cualitativa, se aplicó una **encuesta post-proyecto** ($n = 4$). Para evitar posibles sesgos en los resultados de la

encuesta, el autor del presente trabajo se abstuvo de responder las preguntas de la misma. El **75 %** reportó haber usado la guía durante el desarrollo. En cuanto a experiencia previa, **75 %** se identificó como nivel *básico* y **25 %** como *intermedio*.

1) Horas autorreportadas: Las horas totales autorreportadas (mediana por grupo) sugieren una diferencia a favor del uso de la guía: **con guía** 47 h ($n = 3$) vs. **sin guía** 71 h ($n = 1$). Dado el tamaño muestral reducido, estos valores deben interpretarse con cautela.

2) Adopción de prácticas durante el proyecto (escala 1–5): Se valoró la *frecuencia* de prácticas clave (1=*Nunca*, 5=*Siempre*). Medias por grupo:

- **Documentación continua**: con guía ≈ 2,7, sin guía = 1,0.
- **Pruebas automatizadas**: con guía ≈ 2,7, sin guía = 1,0.
- **Planificación en iteraciones**: con guía = 4,0, sin guía = 4,0.
- **Control de versiones (ramas)**: con guía ≈ 2,3, sin guía = 5,0.

La disparidad en *control de versiones* refleja **heterogeneidad** de prácticas (un caso sin guía reporta uso sistemático de ramas), reforzando la necesidad de estandarización.

3) Percepción específica sobre la guía (solo quienes la usaron; $n = 3$): En la tabla VI se encuentran los resultados de la percepción de los estudiantes que hicieron uso de la guía para su proyecto. Medias de acuerdo (1–5) respecto a afirmaciones sobre la guía:

TABLA VI
PERCEPCIÓN DE LA GUÍA (MEDIA 1–5; $n = 3$).

Afirmación	**Media**
Me ayudó a **planificar** mejor mi trabajo	**5.0**
Me ayudó a **documentar** de forma más clara	**5.0**
Las **recomendaciones** fueron claras y aplicables	**4.67**
Aumentó la **calidad de mis pruebas y validación**	**4.0**
Facilitó escribir **código más mantenible**	**4.0**
Redujo tiempo perdido en tareas repetitivas	**3.67**
Mejoró mis prácticas de **control de versiones**	**3.33**

4) Calidad percibida y usabilidad de la guía: Quienes la usaron calificaron la *claridad/organización* con media **4,33/5** y estimaron un *impacto global en la calidad* de **6,67/10**.

5) Valoración general de la experiencia (todos los participantes): Medias (1–5): *reproducibilidad* 4,5, *claridad de objetivos* 4,75, *organización en fases* 4,5, *confianza en el estado del código* 4,0, *estándares básicos* 4,25. La **productividad comparada** con trabajos previos promedió **4,0/5**.

6) Temas cualitativos de las respuestas abiertas: Utilidad percibida: levantamiento de **requisitos** (SRS), **diseño conceptual** y diagramación como mecanismos para reducir ambigüedad y mejorar la trazabilidad requisito→código.
Dificultades más reportadas: **arquitectura/modularidad** (3 menciones), **planificación/alcance** (2), **pruebas y validación** (2), **documentación** (2), **aprendizaje de herramientas** (2) y **configuración de entorno** (2). Se mencionó el desafío de *abstraer* antes de codificar.
Oportunidades de mejora de la guía: **navegabilidad** (versión más *ligera* para proyectos pequeños), **más ejemplos** por dominio y pautas explícitas para *integración de módulos*, además de refuerzo práctico en **control de versiones** y **pruebas**.

VIII. DISCUSIÓN

Comparación directa: El caso *con guía* muestra que se invierte el 77.9 % del tiempo en **Diseño+Planificación** y registra una **mediana/media** de horas menor frente al caso *sin guía*, lo que sugiere *front-loading* efectivo (reducción de ambigüedad y retrabajo).

Efecto en artefactos y trazabilidad: Las diferencias en rúbrica (89 vs. 58) se explican por: (i) **documentación** y **criterios de aceptación** explícitos, (ii) **organización modular** y (iii) **flujo reproducible**. Sin guía, estos elementos faltan o son débiles, dificultando verificación y mantenimiento.

Triangulación con la percepción de los participantes: La encuesta actualizada respalda que la guía **mejora planificación y documentación** (5.0/5 en ambas) y ofrece **recomendaciones aplicables** (4.67/5). Se perciben **ganancias en pruebas** (4.0/5), aunque la *frecuencia real* de pruebas sigue baja (medias $\approx$ 2,7 con guía vs. 1.0 sin guía). En **control de versiones** el acuerdo es moderado (3.33/5) y se observa **heterogeneidad** en práctica: el caso sin guía reporta uso sistemático de ramas (5/5) frente a 2.3/5 en el grupo con guía. La *claridad/organización* de la guía promedia 4.33/5 y su *impacto global* 6.67/10, con valoración general alta del trabajo (reproducibilidad 4.5/5; organización en fases 4.5/5). En conjunto, los datos cualitativos y cuantitativos convergen en un *efecto positivo* sobre eficiencia estructural y reproducibilidad operativa, con brechas persistentes en **pruebas** y **control de versiones**.

Rendimiento y escalabilidad: No se instrumentaron *benchmarks* sistemáticos; la guía requiere plantillas y protocolos mínimos para medir tiempo/paso y uso de memoria, y recomendaciones de optimización (*e.g.* vectorización/compilación JIT).

Validez y alcance: El tamaño muestral es reducido. Aun así, la **consistencia** entre métricas objetivas (rúbrica, tiempo) y percepciones (planificación, documentación, claridad) refuerza la plausibilidad del *efecto positivo* de la guía en eficiencia estructural y reproducibilidad operativa. Persisten brechas en **pruebas** y **control de versiones**.

IX. RECOMENDACIONES

A. Evaluación y diseño experimental

1. **Ampliar muestra y aleatoriedad:** aumentar n y aleatorizar para reducir sesgos.
2. **Normalizar el registro de horas:** categorías y periodicidad de reporte homogéneas.
3. **Analítica robusta:** tamaños de efecto, ICs y pruebas no paramétricas.

B. Mejora de la guía y de la práctica

1. **Pruebas mínimas obligatorias por dominio:** incluir plantillas (unitarias y *regresión física*) y *criterios de aceptación* verificables. Añadir un *harness* base con ejemplos.
2. **Reproducibilidad verificable:** exigir `requirements.txt`/`environment.yml`, scripts `make`/`tox` y un `README` ejecutable *end-to-end*.
3. **Control de versiones práctico:** módulo corto de **Git manos-a-la-obra** (ramas, `PR`, `merge`/`rebase`) con ejercicios guiados; checklist de flujo (*feature branch* o *trunk-based*).
4. **Benchmarks y perfilamiento:** scripts/notebooks base para medir tiempo/paso y memoria (*escenarios sintéticos*); sugerir `Numba` donde aplique.
5. **Más ejemplos y navegabilidad:** versiones *ligera* y *extendida* de la guía; ejemplos por dominio y **mapa visual** (qué leer según el tamaño del proyecto).
6. **Documentación interna y tipado:** elevar densidad de `docstrings`, ejemplos (`doctest`) y anotaciones `typing`.
7. **Estructura y configuración:** centralizar parámetros físicos; separar ICs, núcleo del solver y visualización.
8. **Retroalimentación iterativa:** actualizar la guía con cada cohorte (*versión viva*).

X. CONCLUSIONES

A. Respecto al objetivo general

La evidencia obtenida indica que la **guía metodológica** mejora, en conjunto, la **eficiencia del proceso** y la **calidad del software** producido por estudiantes de Astronomía. Con guía se observaron: (i) **menor esfuerzo típico** (mediana 44,38 h vs. 57,28 h), (ii) **productos de mayor calidad** (rúbrica 89/100 vs. 58/100; $\Delta = 31$ puntos) y (iii) una **redistribución del trabajo** hacia fases tempranas (Diseño+Planificación 62,7 % vs. 20,6 %). Estos hallazgos convergen en que la guía reduce iteraciones ad hoc, promueve arquitectura modular y eleva la trazabilidad y reproducibilidad de los entregables.

B. Respecto a los objetivos específicos

a) Examinar la guía. La aplicación en dos contextos (estudio de caso personal para N–cuerpos y proyecto SPH externo) muestra que la guía:

- **Fortalece** trazabilidad requisito→código, modularidad/extensibilidad y reproducibilidad operativa (SRS, diseño, empaquetado mínimo y `README`/`requirements`).
- **Induce verificación temprana**, aunque la **adopción de pruebas** aún es insuficiente sin andamiaje adicional; ambos casos evidencian necesidad de ampliar cobertura (*invariantes físicos*, regresión numérica) y **benchmarks** de rendimiento.
- **Limita la ambigüedad** en etapas iniciales y reduce retrabajo; el costo inicial de documentar/diagramar se compensa en integración y mantenimiento.

b) Diseñar el experimento computacional a realizar. El proyecto de *análisis de estabilidad de galaxias de disco* resultó un **banco de prueba pertinente y exigente** (múltiples módulos/algoritmos y decisiones de arquitectura), suficiente para tensionar diseño, validación y rendimiento. La experiencia mostró que la guía facilita organizar el flujo *ingesta→análisis→salidas* y la estandarización de parámetros, mejorando la comparabilidad entre implementaciones.

c) Diseñar el experimento para medir el impacto de la guía. El diseño cuasi–experimental (*control* vs. *experimental*) con **triangulación de instrumentos** (rúbrica

técnica, registro de horas y encuesta) fue **adecuado para detectar efectos** de magnitud media–alta. Amenazas a la validez: tamaño muestral reducido ($n = 2 + 2$), no aleatorización, horas autorreportadas y evaluaciones parciales. Aun así, se observaron diferencias consistentes y en la misma dirección en métricas objetivas y percepciones.

d) Medir la contribución de la guía al proceso de desarrollo de software científico.

- **Eficiencia del proceso**: menor esfuerzo típico y **menor dispersión** en el grupo con guía (SD 3,71 h vs. 19,25 h). Diferencia de medias de +12,91 h a favor del uso de la guía (+29,1 % menos), con $\delta_{\text{Cliff}} \approx 0{,}50$ (efecto *mediano*).
- **Distribución del esfuerzo**: la guía **reorienta el tiempo** hacia Diseño+Planificación (62.7
- **Calidad del producto** (rúbrica): mejora amplia y consistente en *documentación* (+10 pts), *organización* (+6), *funcionalidad* (+6) y *rendimiento* (+7); también en *mantenibilidad* (+2).
- **Percepción de los participantes**: quienes usaron la guía reportaron beneficios altos en *planificación* y *documentación* (medias 5,0/5), y mejora percibida en *pruebas* y *mantenibilidad* ($\approx$ 4,0/5). El *impacto global* percibido se estimó en 6,67/10.
- **Estudios de caso**: el N–cuerpos (86/100) evidencia que la guía impulsa arquitectura clara y reproducibilidad; el SPH (76/100) muestra buena organización y diseño, pero subraya que **sin soporte explícito** los componentes de *verificación/benchmarks* tienden a quedar rezagados.

C. Síntesis global

La guía es **útil, factible y pertinente** para formación en software científico: **reduce esfuerzo típico**, **mejora calidad** y **estabiliza el proceso** al estructurar fases tempranas. Para maximizar su efecto, conviene complementarla con **andamiaje práctico** (plantillas de `pytest` con invariantes físicos, `CI` mínima, cuadernos de *benchmarks* base y ejemplos por dominio). Aun con una muestra pequeña, la **dirección y magnitud** de los efectos observados respaldan la **adopción curricular** de la guía, con énfasis en verificación y medición de rendimiento para cerrar las brechas identificadas.

REFERENCIAS

[1] D. Modak and C. Rahaman, *Research in Image Processing for Medical Applications Using the Secure Smart Healthcare Technique*, 03 2025, pp. 115–153.

[2] G. Wilson, D. Aruliah, C. Brown, N. Chue Hong, M. Davis, R. Guy, S. Haddock, K. Huff, I. Mitchell, M. Plumbley *et al.*, "Best practices for scientific computing," *PLoS Biology*, vol. 12, no. 1, p. e1001745, 2014.

[3] V. Goel and A. Kaur, "A Novel Technique for Optimization of Artificial Neural Network Using Ensemble of Chimp, Harris Hawks and Manta Ray Foraging Optimization Algorithms for Enhancing Software Maintainability Prediction," *Arabian Journal for Science and Engineering*, pp. 1–35, 03 2025.

[4] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE)*. Vancouver, BC, Canada: IEEE, 2009, pp. 1–8.

[5] D. C. Ince, L. Hatton, and J. Graham-Cumming, "The case for open computer programs," *Nature*, vol. 482, no. 7386, pp. 485–488, 2012.

[6] T. Storer, "Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming," *ACM Computing Surveys*, vol. 50, no. 4, Aug. 2017.

[7] A. Connolly, J. Hellerstein, N. Alterman, D. Beck, R. Fatland, E. Lazowska, V. Mandava, and S. Stone, "Software Engineering Practices in Academia: Promoting the 3Rs—Readability, Resilience, and Reuse," *Harvard Data Science Review*, vol. 5, no. 2, 2023.

[8] A. Johanson and W. Hasselbring, "Software Engineering for Computational Science: Past, Present, Future," *Computing in Science & Engineering*, vol. 20, no. 2, pp. 90–109, 2018.

[9] L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott, "Troubling trends in scientific software use," *Science*, vol. 340, no. 6134, pp. 814–815, 2013.

[10] A. Rokem, "Ten simple rules for scientific code review," *PLOS Computational Biology*, vol. 20, no. 9, pp. 1–8, 09 2024.

[11] M. Barker, N. Chue Hong, D. S. Katz *et al.*, "Introducing the FAIR Principles for Research Software," *Scientific Data*, vol. 9, no. 622, 2022.

[12] M. A. Heroux, L. C. McInnes, D. E. Bernholdt, A. Dubey, B. Hadri, J. C. Bennett *et al.*, "Advancing scientific productivity through better scientific software: Developer productivity and software sustainability report," U.S. Department of Energy, Tech. Rep., 2020.

[13] IDEAS Productivity Project, "Productivity and Sustainability Improvement Planning (PSIP) Overview," https://betterscientificsoftware.github.io/PSIP-Tools/PSIP-Overview.html, 2024.

[14] J. Segal, "When software engineers met research scientists: A case study," *Empirical Software Engineering*, vol. 10, no. 4, pp. 517–536, 2005.

[15] Z. Hassan, C. Treude, M. Norrish, G. Williams, and A. Potanin, "Characterising reproducibility debt in scientific software: A systematic literature review," *Journal of Systems and Software*, vol. 222, p. 112327, 2025.

[16] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York: McGraw-Hill Education, 2014.

[17] R. D. Peng, "Reproducible research in computational science," *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011.

[18] J. W. Thomas, "A review of research on project-based learning," Autodesk Foundation, Tech. Rep., 2000.

[19] B. J. S. Barron, D. L. Schwartz, N. J. Vye, A. Moore, A. Petrosino, L. Zech, and J. D. Bransford, "Doing with understanding: Lessons from research on problem- and project-based learning," *Journal of the Learning Sciences*, vol. 7, no. 3-4, pp. 271–311, 1998.

[20] L. F. Capretz, F. Ahmed, and F. Q. da Silva, "Soft skills and software development: A reflection from the software industry," *Computing in Science & Engineering*, vol. 20, no. 6, pp. 6–12, 2018.

[21] I. A. Cosden, K. McHenry, and D. S. Katz, "Research Software Engineers: Career Entry Points and Training Gaps," *Computing in Science & Engineering*, 2023.

[22] L. P. Coelho, "For long-term sustainable software in bioinformatics," *PLOS Computational Biology*, vol. 20, no. 3, p. e1011920, 2024.

[23] F. E. Kohrs *et al.*, "Eleven strategies for making reproducible research and software training part of the curriculum," *eLife*, vol. 12, p. e89736, 2023.

[24] S. Nocera, S. Romano, R. Francese, and G. Scanniello, "Software engineering education: Results from a training intervention based on SonarCloud when developing web apps," *Journal of Systems and Software*, 2025.

[25] *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*, International Organization for Standardization Std. ISO/IEC 25 010:2023, 2023. [Online]. Available: https://www.iso.org/standard/78176.html

[26] *Systems and software engineering — Life cycle processes — Requirements engineering*, ISO/IEC/IEEE Std. ISO/IEC/IEEE 29 148:2018, 2018. [Online]. Available: https://www.iso.org/standard/72089.html

[27] J. H. Davis, P. Sivaraman, J. Kitson, K. Parasyris, H. Menon, I. Minn, G. Georgakoudis, and A. Bhatele, "Taking GPU Programming Models to Task for

Performance Portability," in *Proceedings of the 2025 International Conference on Supercomputing (ICS '25)*. Salt Lake City, UT, USA: ACM, 2025, also available as arXiv:2402.08950. [Online]. Available: https://www.cs.umd.edu/~bhatele/pubs/pdf/2025/ics2025.pdf

[28] C. Melone and S. Jones, "Verifying Functionality and Performance of HPC Applications with Continuous Integration," in *PEARC '23: Practice and Experience in Advanced Research Computing*. ACM, 2023, pp. 460–462.

[29] L. S. Jr., "The Dynamics of the Interstellar Medium. III. Galactic Distribution," *The Astrophysical Journal*, vol. 95, pp. 329–344, 1942.

[30] P. C. van der Kruit and K. C. Freeman, "Galaxy Disks," *Annual Review of Astronomy and Astrophysics*, vol. 49, pp. 301–371, 2011.

[31] J. Binney and S. Tremaine, *Galactic Dynamics, Second Edition*, ser. Princeton Series in Astrophysics. Princeton University Press, 2008.

[32] L. Hernquist, "An Analytical Model for Spherical Galaxies and Bulges," *The Astrophysical Journal*, vol. 356, pp. 359–364, 1990.

[33] J. F. Navarro, C. S. Frenk, and S. D. M. White, "A Universal Density Profile from Hierarchical Clustering," *The Astrophysical Journal*, vol. 490, pp. 493–508, 1997.

[34] M. Cappellari, "Measuring the Inclination and Mass-to-Light Ratio of Axisymmetric Galaxies via Anisotropic Jeans Models of Stellar Kinematics," *Monthly Notices of the Royal Astronomical Society*, vol. 390, no. 1, pp. 71–86, 2008.

[35] A. Toomre, "On the Gravitational Stability of a Disk of Stars," *The Astrophysical Journal*, vol. 139, pp. 1217–1238, 1964.

ANEXOS

La Tabla VII da acceso a los entregables enviados por los participantes en el presente documento.

TABLA VII

ENLACES A ENTREGABLES.

Grupo	**Participante**	**Evidencia (Entregables)**
Caso Piloto	Autor del documento	[Enlace entrega - Autor del Documento]
Experimental	Estudiante Con Guía A	[Enlace entrega - Estudiante Con Guía A]
Experimental	Estudiante Con Guía B	[Enlace entrega - Estudiante Con Guía B]
Control	Estudiante Sin Guía A	[Enlace entrega - Estudiante Sin Guía A]
Control	Estudiante Sin Guía B	[Enlace entrega - Estudiante Sin Guía B]