

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“DISEÑO DE RÚBRICAS PARA LA EVALUACIÓN DE ERRORES CONCEPTUALES EN UN CURSO INTRODUCTORIO DE PROGRAMACIÓN.”

MAXIMILIANO ANDRÉS JOHNSON MORENO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesora Correferente: Andrea Vásquez

Julio - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Diseño de Rúbricas para la Evaluación de Errores Conceptuales en un Curso Introductorio de Programación

Nombre del candidato(a): Maximiliano Andrés Johnson Moreno

Carrera / Grado: Ingeniería Civil Informática

Campus: Valparaíso

Departamento: Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Federico Meza, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 10 de julio de 2026

Firma: 

Estudiante o Candidato(a):

Fecha: 10 de julio de 2026

Firma: 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A María del Carmen Moreno, Claudia Moreno, Marcelo Johnson y Aileen Riquelme.

AGRADECIMIENTOS

Agradezco a todas las personas que me acompañaron durante mi período universitario. Gracias a mi mamá y a mi papá por apoyarme en todo; también a Gerald y a mi tía Claudia, que siempre estuvieron presentes, y a mi Mimi, que me dio muchas herramientas que siempre usaré.

Gracias a mi grupo de amigos, ya que con ellos el tiempo en la universidad pasó muy rápido, con muy buenos momentos juntos. También le agradezco a mi pareja, que me dio la fuerza para seguir adelante en muchos días complicados e hizo que me diera cuenta de lo que realmente soy capaz.

Por último, le agradezco a mi profesor guía, Federico Meza, por acompañarme y guiarme todo este último semestre y siempre responder de la mejor manera. También le agradezco a la profesora Andrea Vásquez por ayudarme en puntos importantes de mi trabajo.

RESUMEN

Resumen— Este trabajo presenta el diseño y validación de un instrumento de diagnóstico temprano y una rúbrica analítico-cognitiva para detectar errores conceptuales (*misconceptions*) en programación básica. Mediante un Inventario de Conceptos aplicado a estudiantes de ingeniería ($N = 133$ y $N = 103$), se clasificaron las fallas en tres niveles de severidad utilizando un Criterio de Gravedad Máxima. Los resultados revelaron deficiencias estructurales invisibles para la evaluación tradicional, destacando un 82,5 % de error crítico en el quiebre de flujo (`return`) y un 40,6 % en enrutamiento (`if` vs `while`). Un sondeo docente validó que visualizar esta información en un Panel Analítico facilita las intervenciones pedagógicas preventivas. Se concluye que este triaje cognitivo fomenta un modelo de evaluación formativo y escalable, optimizando las acciones remediales antes de las evaluaciones sumativas.

Palabras Clave— Aprendizaje de programación; Errores conceptuales; Rúbrica formativa; Triaje cognitivo; Analítica de aprendizaje.

ABSTRACT

Abstract— This work presents the design and validation of an early diagnostic instrument and an analytical-cognitive rubric to detect conceptual errors (*misconceptions*) in introductory programming. Through a Concept Inventory applied to engineering students ($N = 133$ and $N = 103$), errors were classified into three severity levels using a Maximum Severity Criterion. Results revealed structural flaws invisible to traditional assessment, highlighting an 82.5% critical error rate in flow interruption (`return`) and 40.6% in routing (`if` vs `while`). A qualitative teacher survey validated that visualizing this data in an Analytical Dashboard facilitates targeted preventive interventions. It is concluded that this cognitive triage fosters a scalable and formative assessment model, optimizing remedial actions prior to summative exams.

Keywords— Programming learning; Misconceptions; Formative rubric; Cognitive triage; Learning analytics.

GLOSARIO

COVID-19: *Coronavirus Disease 2019*.

DI: Departamento de Informática.

LMS: *Learning Management System* (Sistema de Gestión de Aprendizaje).

LTI: *Learning Tools Interoperability* (Interoperabilidad de Herramientas de Aprendizaje).

MC³: *Misconceptions in Correct Code* (Problemas de Comprensión en Códigos Correctos).

OE: Objetivo Específico.

P1 a P10: Pregunta 1 a Pregunta 10 (Abreviatura de los reactivos del instrumento).

UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto y Situación Actual	3
1.2 Actores Involucrados y sus Dificultades	3
1.3 Impacto y Consecuencias del Problema	4
1.4 Análisis de Soluciones Existentes y Justificación	4
1.5 Objetivos y Alcances de la Memoria	5
1.5.1 Objetivo General	5
1.5.2 Objetivos Específicos	5
CAPÍTULO 2: MARCO CONCEPTUAL	6
2.1 Modelos de Comprensión en la Programación	6
2.1.1 Modelos Mentales vs. Modelos Conceptuales	6
2.1.2 La Máquina Nocional (<i>Notional Machine</i>): Definición y Propósito . . .	6
2.1.3 Representaciones y Ejemplos de la Máquina Nocional	7
2.2 Errores en el Aprendizaje de la Programación	7
2.2.1 Dimensiones del Conocimiento: Sintáctico, Conceptual y Estratégico .	7
2.2.2 Misconceptions	7
2.3 Desafíos Cognitivos y <i>Misconceptions</i> Específicos en Python	8
2.3.1 Tipado Dinámico y Sobrecarga de Operadores	8
2.3.2 El Paradigma de "Todo es una Referencia"	8
2.4 Métodos de Detección y Diagnóstico de Errores	9
2.4.1 Pruebas de Escritorio (<i>Tracing</i> o Ruteo)	9
2.4.2 Inventarios de Conceptos (<i>Concept Inventories</i>)	9
2.4.3 Protocolos de Pensamiento en Voz Alta	9
2.5 Evaluación mediante Rúbricas en Programación	10
2.5.1 De la evaluación punitiva al diagnóstico cualitativo	10
2.5.2 Rúbricas analíticas enfocadas en síntomas observables	10
2.5.3 Criterio de Gravedad Máxima y Triage Cognitivo	10
2.6 Síntesis Conceptual y Flujo de Intervención	11
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	13

3.1	Población y Muestra	13
3.2	Diseño del Instrumento de Diagnóstico Temprano	13
3.2.1	Diseño Modular y Mitigación de la Fatiga Cognitiva	14
3.2.2	Batería de Preguntas del Instrumento	14
3.3	Validación Preliminar de Constructo y Refinamiento	20
3.3.1	Validación Preliminar de Constructo	20
3.3.2	Mitigación de la Fatiga de Prueba y Preguntas Descartadas	20
3.4	Diseño de la Rúbrica Formativa Específica	22
3.4.1	Regla de Triage Cognitivo y Criterio de Gravedad Máxima	22
3.5	Matriz Diagnóstica - Fase 1: Fundamentos (Forma A)	23
3.6	Matriz Diagnóstica - Fase 2: Abstracción (Forma A)	24
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		27
4.1	Descripción del Escenario de Validación	27
4.2	Resultados de la Fase 1: Fundamentos de Programación	28
4.2.1	Tratamiento de Respuestas Omitidas	28
4.2.2	Análisis Cuantitativo del Diagnóstico	28
4.2.3	Discusión de los Resultados y Observaciones Teóricas	28
4.3	Resultados de la Fase 2: Abstracción y Modularidad	29
4.3.1	Análisis Cuantitativo del Diagnóstico	29
4.3.2	Discusión de los Resultados y Observaciones Teóricas	29
4.4	Evaluación del Potencial Diagnóstico para la Gestión Docente	30
4.4.1	Generación del Panel Analítico (Dashboard Docente)	30
4.5	Sondeo Exploratorio de Percepción Docente	35
4.6	Limitaciones del Estudio	36
CAPÍTULO 5: CONCLUSIONES		38
5.1	Identificación y categorización de <i>misconceptions</i>	38
5.2	Diseño y resultados empíricos de la rúbrica determinista	38
5.3	Impacto del diagnóstico en la gestión docente	39
5.4	Limitaciones del estudio	39
5.5	Trabajo futuro	40
5.6	Ampliación a nuevas unidades temáticas	40
5.7	Integración en plataformas de evaluación continua	40
5.8	Medición del impacto longitudinal en los modelos mentales	40
ANEXOS		41
REFERENCIAS BIBLIOGRÁFICAS		51

ÍNDICE DE FIGURAS

1	Diagrama de flujo conceptual: Detección, triaje y mitigación de un <i>misconception</i> a través de la propuesta metodológica.	12
2	Distribución porcentual de los modelos mentales por reactivo. Se observa la prevalencia de errores estructurales (Nivel 1) en las preguntas P3 y P10. . . .	32
3	Distribución de modelos mentales en la Pregunta 10 (<i>return</i> en ciclos). Representa el colapso cognitivo más severo del instrumento (82,5 % de error crítico).	33
4	Distribución de modelos mentales en la Pregunta 3 (<i>if</i> vs <i>while</i>). Destaca un 40,6 % de colapso de enrutamiento.	33
5	Distribución de modelos mentales en la Pregunta 6 (Diccionarios). Se observa una fuerte fragmentación y dependencia de ciclos (Nivel 2).	34
6	Distribución de modelos mentales en la Pregunta 7 (Referencias en listas). Evidencia una polarización casi simétrica entre dominio y falla estructural. . .	34

ÍNDICE DE TABLAS

1	Rúbrica granular de 3 niveles para la Fase 1. Mapeo correspondiente a la Forma A.	23
2	Rúbrica granular de 3 niveles para la Fase 2. Mapeo correspondiente a la Forma A.	25
3	Frecuencia y clasificación cognitiva de las respuestas obtenidas en la Fase 1 ($N = 133$). Los datos han sido homologados a la Forma A del instrumento. .	28
4	Frecuencia y clasificación cognitiva de las respuestas obtenidas en la Fase 2 ($N = 103$). Los datos han sido homologados a la Forma A del instrumento. .	29
5	Dashboard analítico consolidado para el docente. Fase 1 ($N = 133$) y Fase 2 ($N = 103$).	31

INTRODUCCIÓN

El aprendizaje de la programación en etapas introductorias universitarias se ha consolidado como un pilar fundamental en la formación de estudiantes de ingeniería. Sin embargo, dominar la lógica computacional conlleva una carga cognitiva excepcionalmente alta. En su primer acercamiento, los estudiantes novatos no solo deben aprender la sintaxis estricta de un lenguaje artificial, sino que deben desarrollar una abstracción mental precisa sobre cómo la computadora procesa y almacena la información.

En este proceso de asimilación, es altamente frecuente que la intuición humana y la transferencia de conocimientos matemáticos previos generen esquemas mentales defectuosos. Estas deficiencias, conocidas en la literatura de educación informática como *misconceptions* (errores conceptuales profundos), provocan que el estudiante construya un modelo no viable de la “máquina nocional” subyacente. El problema central que motiva esta investigación radica en que los sistemas de evaluación tradicionales en las asignaturas de programación suelen operar bajo un paradigma sumativo y punitivo, enfocado puramente en la correctitud sintáctica o en que el código final funcione. Este enfoque actúa como una caja negra que califica el resultado, pero enmascara las deficiencias estructurales —fenómeno documentado como Problemas de Comprensión en Códigos Correctos (MC³)—, permitiendo que alumnos con modelos mentales rotos avancen en el curso a través de la memorización o el ensayo y error, para luego sufrir colapsos cognitivos insalvables en evaluaciones de mayor complejidad.

Frente a la masificación de las aulas y la dificultad del equipo docente para brindar retroalimentación personalizada a tiempo, surge la necesidad de transitar hacia un modelo de diagnóstico formativo y automatizable. En este contexto, la presente memoria propone el diseño, aplicación y validación de una metodología de alerta temprana. La solución desarrollada consiste en la creación de un Inventario de Conceptos modular (reactivos diagnósticos conocidos coloquialmente como “Caza Bobos”), cuyos distractores están diseñados empíricamente para capturar errores conceptuales específicos en el lenguaje Python.

A diferencia de las pautas de corrección estándar, la metodología empleada en este trabajo procesa los resultados a través de una Rúbrica Analítico-Cognitiva de carácter determinista. Operando bajo el principio psicométrico del Triage Cognitivo y el Criterio de Gravedad Máxima, la rúbrica clasifica las respuestas de los estudiantes en tres niveles clínicos (Error Conceptual, Error Superficial y Logro Esperado), aislando el problema de raíz y evitando que los aciertos mecánicos oculten la incomprensión de constructos críticos como el control de flujo, el modelo de memoria o el encapsulamiento.

Para validar el impacto y la aplicabilidad de esta propuesta, la investigación se desarrolló metodológicamente a través de un estudio empírico mixto. Cuantitativamente, el instrumento fue aplicado en dos fases modulares a cohortes de primer año de ingeniería de la Universidad Técnica Federico Santa María (UTFSM). Los datos recabados fueron procesados y traducidos

a un Panel Analítico diseñado para la gestión docente. Cualitativamente, mediante un sondeo exploratorio, se evaluó la percepción de profesores titulares respecto a la utilidad de estas analíticas visuales para la toma de decisiones y la replanificación de estrategias pedagógicas preventivas.

Para detallar el desarrollo de esta investigación, el presente documento se estructura en cinco capítulos:

- El **Capítulo 1** define la naturaleza y alcance del problema abordado, contextualizando las dificultades a las que se enfrentan tanto los estudiantes novatos como el equipo docente, justificando la necesidad de la solución propuesta y estableciendo los objetivos de la memoria.
- El **Capítulo 2** expone el marco conceptual que sustenta la investigación. Se profundiza en la teoría de los modelos mentales, la máquina nocional y la literatura contemporánea sobre los *misconceptions* específicos que genera la arquitectura del lenguaje Python.
- El **Capítulo 3** detalla el diseño metodológico de la propuesta de solución. Se describe la justificación del diseño modular del instrumento para evitar la fatiga cognitiva, la creación de la batería de preguntas y la formulación estricta de las reglas de la Rúbrica Formativa Determinista.
- El **Capítulo 4** presenta la validación empírica de la herramienta. Se analizan cuantitativamente los resultados extraídos de la aplicación en el aula (divididos en fase de fundamentos y de abstracción) y se expone el estudio cualitativo que valida la utilidad del Panel Analítico a través de la retroalimentación del equipo docente.
- Finalmente, el **Capítulo 5** consolida las conclusiones del trabajo, contrastando los hallazgos con los objetivos planteados inicialmente, y propone líneas de investigación para el trabajo futuro institucional.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto y Situación Actual

El aprendizaje de lenguajes de programación en cursos introductorios a la informática se presenta como un desafío cognitivo significativo para los estudiantes de ingeniería [du Boulay, 1986]. En ciertos escenarios académicos, esta dificultad se agrava dado que no todos los alumnos ingresan con la predisposición de aprender a programar, o perciben estas asignaturas como elementos ajenos a su área principal de estudio. La evidencia en la literatura de educación informática señala que, durante este proceso, los estudiantes novatos suelen desarrollar *misconceptions* (errores conceptuales profundos), los cuales obstaculizan su progreso y comprensión del funcionamiento lógico de los programas [Qian y Lehman, 2017, Pea, 1986].

El marco teórico respaldado por diversos autores [Sorva, 2012, Fincher *et al.*, 2020] sostiene que gran parte de estos errores conceptuales surgen porque los estudiantes no logran construir un modelo mental viable de la "máquina nocional" (*notional machine*) [Boulay, 1981]. En síntesis, el estudiante no comprende la abstracción subyacente del sistema al ejecutar un código. Como resultado, al enfrentarse a las evaluaciones sumativas, los alumnos reflejan un rendimiento deficiente, y la retroalimentación que reciben en esa etapa resulta tardía o superficial, impidiendo que corrijan la raíz de su modelo mental erróneo a tiempo.

1.2. Actores Involucrados y sus Dificultades

El problema planteado afecta directamente a dos grupos de actores dentro del ecosistema educativo del curso de Introducción a la Programación en la Universidad Técnica Federico Santa María (UTFSM):

1. Estudiantes de primer año: Presentan serias dificultades para comprender la ejecución del código. Estas deficiencias emanan de la transferencia negativa de conocimientos desde el álgebra [Bonar y Soloway, 1985], la incapacidad para razonar sobre el estado dinámico del programa [Sirkiä y Sorva, 2012], y la falta de práctica en la lectura y simulación mental del código.
2. Equipo docente: Los profesores y ayudantes encargados de la evaluación enfrentan el desafío de diagnosticar estos errores a gran escala. A menudo, la corrección automatizada o las pautas rígidas terminan ocultando la raíz cognitiva de los verdaderos errores del estudiante, evaluando únicamente el resultado final y no el proceso lógico subyacente [da Silva, 2024].

1.3. Impacto y Consecuencias del Problema

No abordar esta problemática de manera temprana causará que los alumnos avancen en su malla académica sin poseer una base algorítmica sólida. Esta deficiencia estructural genera fricciones al momento de enfrentarse a cursos más avanzados. Ante la frustración de percibir la programación como una caja negra inaccesible, tal como lo define Haberman [Haberman y Ben-David Kolikant, 2001], los estudiantes suelen recurrir a prácticas de codificación basadas en el ensayo y error, o en la copia de fragmentos de código sin análisis crítico (*copy-paste*), perpetuando así una base de conocimiento frágil.

1.4. Análisis de Soluciones Existentes y Justificación

Actualmente, el sistema de evaluación tradicional mediante rúbricas posee un enfoque sumativo y punitivo: se limita a contabilizar la cantidad de errores visibles al final de un ciclo académico, agrupando problemas de distinta naturaleza semántica y cognitiva en una misma penalización. Esta carencia diagnóstica impide detectar a tiempo si un estudiante sufre de un *misconception* particular antes de rendir un certamen.

Dentro de la UTFSM, han existido investigaciones previas para abordar estas falencias. Vásquez Pinto [Vásquez Pinto, 2020] propuso una rúbrica cualitativa enfocada en la explicación oral de los programas. Sin embargo, dicha investigación presentó problemas de escalabilidad: elicitación aproximaciones de la estructura cognitiva revisando explicaciones individuales no es viable para cursos masivos [Ben-Ari, 1998], sesgando la muestra hacia estudiantes con mayor facilidad comunicativa. Por otro lado, iniciativas recientes como la de Alvayay [Alvayay Figueroa, 2024] exploraron la mitigación de errores a través del ruteo mental en videojuegos. Esto confirma empíricamente la recomendación del estudio multinacional de Lister *et al.* [Lister *et al.*, 2004], demostrando que observar a los novatos rastrear el código linealmente (*concrete tracing*) es la manera más efectiva de evidenciar sus errores conceptuales.

Considerando el retorno a la presencialidad, es imperativo contar con un enfoque preventivo que no dependa exclusivamente de explicaciones orales ni de correctores automáticos que invisibilizan el error humano [da Silva, 2024]. Por ello, la solución propuesta en esta memoria separa el problema en dos componentes complementarios: primero, la aplicación de un cuestionario diagnóstico estructurado para forzar la exposición del modelo mental; y segundo, una rúbrica analítico-cognitiva que procese dichos resultados para perfilar a los estudiantes en distintos niveles de riesgo, proveyendo al docente de información accionable antes de las evaluaciones definitivas.

1.5. Objetivos y Alcances de la Memoria

1.5.1. Objetivo General

Validar una rúbrica formativa y analítico-cognitiva, acoplada a un instrumento de diagnóstico temprano, para el triaje de errores conceptuales (*misconceptions*) en estudiantes novatos de Programación de la UTFSM, con el fin de proveer un perfil preventivo que apoye la toma de decisiones docentes antes de las evaluaciones sumativas.

1.5.2. Objetivos Específicos

- Fundamentar y operacionalizar las categorías de *misconceptions* más frecuentes en la enseñanza introductoria de programación, basándose en la literatura especializada y el contexto local de la UTFSM.
- Diseñar un instrumento de diagnóstico temprano y una rúbrica analítico-cognitiva estructurada en distintos niveles de riesgo, orientados a detectar y clasificar las deficiencias en la máquina nocional del estudiante.
- Evaluar la utilidad de la rúbrica como sistema de alerta temprana, validando su capacidad para proveer al equipo docente de información oportuna que oriente intervenciones pedagógicas preventivas.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Modelos de Comprensión en la Programación

Existe un dilema intrínseco entre la relación del lenguaje artificial —entendido por las máquinas— y el lenguaje natural, el cual es utilizado por los seres humanos. Esta brecha genera conflictos significativos al momento de aprender y enseñar programación. Un estudiante novato tiende a “hablar con el programa” debido a una base escasa de conocimientos técnicos, lo que ocasiona errores sintácticos y la formación de modelos mentales erróneos.

2.1.1. Modelos Mentales vs. Modelos Conceptuales

El modelo mental se define como un esquema interno utilizado para representar relaciones entre objetos o representaciones de la realidad, describiendo su funcionamiento [Justi y Van Driel, 2006]. Por otro lado, el modelo conceptual es una representación abstracta, unívoca y simplificada de objetos y sus relaciones en la realidad.

En el ámbito educativo, los profesores enseñan modelos conceptuales y los estudiantes construyen sus propios modelos mentales, los cuales pueden ser más o menos similares al modelo explicado [Gallego-Durán *et al.*, nd]. Es en este punto donde la labor docente cobra relevancia, ya que la forma de explicar un constructo (como un ciclo `while`) influye directamente en la solidez del esquema mental del alumno. Muchos novatos construyen estos modelos sobre la marcha mientras programan, lo que frecuentemente deriva en interpretaciones incorrectas del flujo de ejecución¹.

2.1.2. La Máquina Nocional (*Notional Machine*): Definición y Propósito

Un agente crítico en la intersección de los modelos mentales y conceptuales es la máquina nocional (*Notional Machine*). Aunque no existe una definición única, la más aceptada es la de du Boulay: “A *notional machine* is the idealized model of the computer implied by the constructs of the programming language” [du Boulay, 1986]. En esencia, es un modelo conceptual diseñado para enseñar cómo funciona un lenguaje de programación específico [Fincher *et al.*, 2020].

Este modelo no se encasilla en las limitaciones físicas reales de un sistema informático ni se liga exclusivamente al modelo mental del estudiante; puede explicarse mediante texto,

¹Un modelo mental incorrecto puede ser persistente y difícil de corregir si no se detecta a tiempo mediante una intervención pedagógica adecuada.

diagramas o ruteos [Iturbide, 2021]. Sorva [Sorva, 2012] aboga por la Simulación Visual de Programas, donde el estudiante asume el rol de la computadora en una actividad de “lectura y rastreo” (*tracing*), utilizando visualizaciones interactivas para ejecutar el código paso a paso. El propósito es extraer los elementos no visibles de la “caja negra” del lenguaje para que el alumno construya un modelo mental sólido.

2.1.3. Representaciones y Ejemplos de la Máquina Nocional

Las representaciones pueden clasificarse en tres categorías:

1. **Mentales:** Simulación abstracta y privada del código en el cerebro del estudiante.
2. **Explícitas no ejecutables:** Diagramas de flujo, ruteos en papel o dibujos en pizarras.
3. **Generadas por software:** Herramientas automatizadas que ilustran la ejecución en tiempo real, como *Python Tutor*, lenguajes de bloques (*Scratch*) o simuladores de microprocesadores.

2.2. Errores en el Aprendizaje de la Programación

2.2.1. Dimensiones del Conocimiento: Sintáctico, Conceptual y Estratégico

Se propone dividir el conocimiento de programación en tres dimensiones fundamentales [Qian y Lehman, 2017]:

- **Sintáctico:** Dominio de las reglas gramaticales y hechos básicos de un lenguaje específico.
- **Conceptual:** Comprensión de los principios y constructos; es el entendimiento de qué ocurre dentro de la computadora. Aquí es donde surgen los *misconceptions* o errores conceptuales profundos.
- **Estratégico:** Habilidad de aplicar los conocimientos previos para resolver problemas nuevos, planificar, escribir y depurar programas.

2.2.2. Misconceptions

El pensamiento computacional requiere pensar en distintos niveles de abstracción. El término *misconception* hace referencia a una idea equívoca que las personas tienen sobre un

tema, resultando en interpretaciones deficientes para contextos prácticos [Sorva, 2012]. En la literatura, se identifican diversas causas para estos errores, que van desde la interferencia del lenguaje natural hasta la aplicación incorrecta de conocimientos matemáticos previos.

2.3. Desafíos Cognitivos y *Misconceptions* Específicos en Python

Si bien lenguajes como Python han ganado gran popularidad en los cursos introductorios de programación debido a su sintaxis amigable y baja barrera de entrada, la literatura advierte que es común confundir esta “simplicidad sintáctica” con “simplicidad semántica” [Sorva, 2012, da Silva, 2024]. Por diseño arquitectónico, la sintaxis minimalista de Python oculta una máquina nociónal sumamente compleja, lo que propicia que los estudiantes desarrollen modelos mentales frágiles, inadecuados y propensos a la ruptura [da Silva, 2024]. A diferencia de lenguajes tradicionales de tipado estático como C++ o Java, el diseño subyacente de Python presenta desafíos cognitivos únicos que actúan como fuentes primarias de *misconceptions*:

2.3.1. Tipado Dinámico y Sobrecarga de Operadores

En Python, el tipado dinámico exige al estudiante de declarar explícitamente los tipos de datos en la memoria. Sumado a esto, el lenguaje hace un uso intensivo de la sobrecarga de operadores (*overloading*)² [Sorva, 2012]. Por ejemplo, el operador + o la asignación incrementada += utilizan exactamente la misma sintaxis tanto para operaciones aritméticas simples como para la mutación y concatenación de estructuras de datos complejas (como listas o cadenas) [Qian y Lehman, 2017]. Esta sobrecarga sintáctica oculta la verdadera complejidad de los algoritmos subyacentes, invitando al estudiante novato a aplicar intuiciones matemáticas erróneas sobre tipos de datos que se comportan de manera totalmente distinta [Qian y Lehman, 2017, da Silva, 2024]. Diversos estudios sugieren que la enseñanza exclusiva de Python en cursos introductorios, sin un andamiaje que explique la máquina nociónal subyacente, a menudo deja a los alumnos con deficiencias profundas en la comprensión del paso de parámetros, la compilación y los punteros; conceptos que en lenguajes de más bajo nivel como C son visibilizados de manera explícita [Sorva, 2012].

2.3.2. El Paradigma de “Todo es una Referencia”

A diferencia de lenguajes en los cuales las variables actúan como contenedores físicos que almacenan valores directamente (paso por valor), en Python el concepto de variable ope-

²La sobrecarga de operadores (*overloading*) permite que un mismo símbolo matemático o sintáctico ejecute comportamientos algorítmicos radicalmente distintos dependiendo del tipo de objeto subyacente que lo invoque.

ra universalmente como una referencia dinámica a un objeto [da Silva, 2024]. Este es considerado un “concepto umbral” (*threshold concept*)³ que resulta crítico y altamente problemático para los programadores novatos [Meyer y Land, 2003, Sorva, 2012]. Cuando un estudiante ejecuta una instrucción simple como `lista2 = lista1`, el modelo mental intuitivo sugiere que se ha creado una copia independiente y aislada de la colección de datos [Ma, 2007]. Sin embargo, en la máquina nocional de Python, ambas variables pasan a apuntar (referenciar) exactamente al mismo objeto mutable en la memoria física [Sorva, 2012, Weintrop y Wilensky, 2015]. Comprender la diferencia entre identidad, mutabilidad, creación de copias y compartición de datos impone una carga cognitiva altísima para quien recién comienza. Esto detona errores severos en etapas tempranas al utilizar métodos destructivos como `.append()`, donde el estudiante ignora por completo que la mutación afectará simultáneamente a todas las variables que referencien a dicho objeto [da Silva, 2024].

2.4. Métodos de Detección y Diagnóstico de Errores

2.4.1. Pruebas de Escritorio (*Tracing* o Ruteo)

El ruteo es la acción de realizar un seguimiento manual paso a paso al código [Alvayay Figueroa, 2024]. Aunque es una herramienta diagnóstica útil, exige una alta carga cognitiva para los novatos [Sorva, 2012]. Herramientas como videojuegos educativos han implementado mecánicas de *logs* para facilitar este proceso y corregir errores de lógica de forma interactiva.

2.4.2. Inventarios de Conceptos (*Concept Inventories*)

Son instrumentos de evaluación validados donde las opciones incorrectas están basadas en *misconceptions* documentados empíricamente [da Silva, 2024]. Permiten identificar con precisión qué concepto específico no ha sido comprendido por el estudiante [Kaczmarczyk *et al.*, 2010].

2.4.3. Protocolos de Pensamiento en Voz Alta

Consiste en pedir al alumno que verbalice su proceso de pensamiento mientras resuelve un problema [Kaczmarczyk *et al.*, 2010]. Otra variante es el análisis de explicaciones de código

³Un *concepto umbral* se define como una idea central en una disciplina que, una vez comprendida, transforma irreversiblemente la forma en que el estudiante percibe la materia, pero que resulta excepcionalmente difícil de asimilar en un principio y suele generar bloqueos en el aprendizaje [Meyer y Land, 2003].

a posteriori, evaluando mediante rúbricas si el estudiante utiliza correctamente términos formales [Vásquez Pinto, 2020].

2.5. Evaluación mediante Rúbricas en Programación

2.5.1. De la evaluación punitiva al diagnóstico cualitativo

Tradicionalmente, la evaluación de proyectos en cursos introductorios de programación se ha centrado en enfoques sumativos orientados a la penalización. Las pautas de corrección tradicionales suelen definir los niveles de desempeño mediante el conteo de fallos (ej. “comete 1 o 2 errores”), agrupando en una misma penalización problemas de naturaleza radicalmente distinta, como no inicializar un contador u olvidar un paréntesis.

Este enfoque cuantitativo oculta el verdadero estado cognitivo del estudiante. Como demuestra da Silva [da Silva, 2024], evaluar únicamente basándose en si el código compila puede llevar a que los alumnos obtengan calificaciones aprobatorias mientras siguen albergando *misconceptions* severos sobre cómo funciona la memoria. Para subsanar esto, la educación informática moderna propone aplicar rúbricas analíticas de diagnóstico antes de las evaluaciones sumativas, lo que permite al docente intervenir oportunamente y al estudiante reflexionar sobre las deficiencias en su modelo mental.

2.5.2. Rúbricas analíticas enfocadas en síntomas observables

Dado que los modelos mentales y los errores conceptuales son representaciones internas ocultas, las herramientas de evaluación deben diseñarse bajo un enfoque basado en síntomas (*symptom-based approach*). Esto implica que la rúbrica no evalúa el error abstracto en sí, sino el comportamiento anómalo o las incoherencias observables que dicho error provoca cuando el estudiante interactúa con un fragmento de código específico.

2.5.3. Criterio de Gravedad Máxima y Triage Cognitivo

Para que un instrumento diagnóstico sea verdaderamente útil en la gestión académica masiva, no basta con identificar los errores; es imperativo jerarquizarlos. Por ello, la metodología propuesta en esta memoria se fundamenta en el concepto de “Triage Cognitivo” y la aplicación del “Criterio de Gravedad Máxima”.

En la psicometría educativa enfocada en la programación, promediar linealmente los aciertos mecánicos con las fallas conceptuales genera un peligroso falso positivo de aprendizaje [da Silva, 2024]. Si un estudiante comete un fallo sintáctico o estratégico superficial (clasifi-

cado como Nivel 2), demuestra simplemente una sobrecarga cognitiva temporal propia del novato. Sin embargo, si el alumno marca un distractor correspondiente a un *misconception* documentado (clasificado como Nivel 1 Crítico), evidencia un colapso severo en la abstracción de su máquina nocional [Qian y Lehman, 2017].

Según la literatura especializada [Vásquez Pinto, 2020], un *misconception* actúa como una barrera insalvable que bloquea y paraliza la adquisición futura de conocimientos más complejos. Por lo tanto, el Criterio de Gravedad Máxima establece que es pedagógicamente justificable y necesario clasificar al alumno en base a su error más severo. Si el esquema mental estructural está roto, los aciertos mecánicos periféricos son producto de la intuición o memorización, careciendo de validez asimilativa. Esta regla de precedencia diagnóstica justifica la necesidad de priorizar la acción remedial urgente sobre aquellos estudiantes que presentan fallas de Nivel 1.

Bajo este mismo paradigma de triaje, la omisión de respuesta o la entrega de código completamente inválido frente a un reactivo focalizado no se considera un descuido superficial, sino una manifestación clínica de **colapso de abstracción**. En la psicometría de la programación, la incapacidad de un estudiante para iniciar el ruteo o la estructuración de una solución elemental refleja que la máquina nocional para ese constructo es inexistente o no viable [Sorva, 2012]. Por consiguiente, el *Criterio de Gravedad Máxima* dictamina que las omisiones deben ser categorizadas automáticamente en el Nivel 1 Crítico, garantizando que el estudiante reciba intervención de reconstrucción basal.

2.6. Síntesis Conceptual y Flujo de Intervención

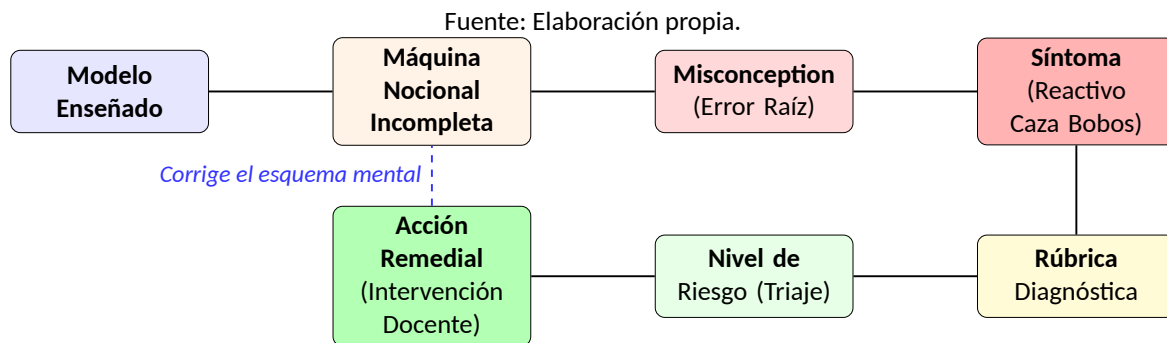
Para asegurar la coherencia técnica a lo largo de este documento, y alineándose con el marco teórico expuesto, se definen explícitamente los cinco pilares conceptuales de esta investigación:

- **Modelo Mental:** Esquema cognitivo interno, privado y altamente maleable que construye un estudiante para representar el funcionamiento subyacente de un sistema de *software* [Justi y Van Driel, 2006].
- **Máquina Nocional (*Notional Machine*):** Modelo conceptual idealizado del funcionamiento interno de la computadora que el lenguaje de programación implica. Es la representación pedagógica formal que el docente busca transferir al alumno [Fincher *et al.*, 2020].
- **Misconception:** Falla estructural profunda. Idea equívoca, persistente y resiliente al cambio sobre un constructo basal de la programación, resultando en un modelo mental no viable que bloquea el aprendizaje posterior [Qian y Lehman, 2017].

- **Triage Cognitivo:** Procedimiento psicométrico de clasificación y priorización de los estudiantes en distintos niveles de riesgo de fracaso académico, basado en la severidad diagnosticada de las deficiencias en su modelo mental.
- **Rúbrica Formativa:** Matriz de evaluación analítico-cognitiva, diseñada no bajo un enfoque punitivo o calificador, sino como un motor de diagnóstico preventivo. Su fin es proveer retroalimentación accionable antes de una evaluación sumativa.

El proceso completo propuesto en esta memoria, desde la inducción teórica fallida hasta la corrección sistemática del error, se sintetiza en el diagrama conceptual de la Figura 1. Este esquema ilustra cómo un concepto abstracto mal asimilado se traduce en un síntoma medible, el cual, al ser interceptado por la rúbrica, gatilla el triaje y la consecuente intervención docente.

Figura 1: Diagrama de flujo conceptual: Detección, triaje y mitigación de un *misconception* a través de la propuesta metodológica.



CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

El presente capítulo detalla el desarrollo de la solución propuesta para el diagnóstico temprano de errores conceptuales en estudiantes de primer año de programación. La esencia de este trabajo radica en la transición de un modelo de evaluación sintáctico y punitivo hacia un instrumento cualitativo basado en rúbricas analítico-cognitivas. A continuación, se expone la población objetivo, la metodología utilizada para el diseño del instrumento modular y la estructuración de la rúbrica de evaluación determinista.

3.1. Población y Muestra

El instrumento de diagnóstico temprano fue diseñado para aplicarse a la población de estudiantes de primer año de ingeniería que cursan la asignatura de Introducción a la Programación en la Universidad Técnica Federico Santa María (UTFSM)⁴.

Para la validación empírica y el ajuste de la rúbrica analítico-cognitiva, se estructuró un diseño de aplicación modular. Se trabajó con una muestra no probabilística por conveniencia extraída de cursos paralelos dictados tanto en el Campus Santiago (San Joaquín) como en la Casa Central (Valparaíso), mitigando posibles sesgos demográficos o de docencia local.

Dado el carácter modular del instrumento, la recolección de datos arrojó dos tamaños muestrales distintos:

- **Fase 1 (Control de Flujo y Modelo de Memoria, Preguntas 1 a 5):** Se obtuvo una muestra válida de $N = 133$ estudiantes.
- **Fase 2 (Estructuras de Datos y Scope, Preguntas 6 a 10):** Se obtuvo una muestra válida de $N = 103$ estudiantes.

3.2. Diseño del Instrumento de Diagnóstico Temprano

Para llevar a cabo el diagnóstico del estado cognitivo de los estudiantes antes de las evaluaciones sumativas, se diseñó un instrumento basado en la metodología de Inventarios de Conceptos (*Concept Inventories*). A diferencia de las evaluaciones tradicionales que se centran puramente en la correctitud sintáctica o en la validación mediante casos de prueba, este instrumento adopta un enfoque basado en síntomas observables [da Silva, 2024]. El objetivo es hacer visible el modelo mental que el estudiante posee sobre la “máquina nociónal”

⁴Tanto UTFSM como otras siglas técnicas empleadas a lo largo de este documento se encuentran detalladas en la sección de Glosario.

subyacente del lenguaje de programación.

3.2.1. Diseño Modular y Mitigación de la Fatiga Cognitiva

La aplicación de pruebas de seguimiento de código (*tracing* o ruteo) exige una alta carga en la memoria de trabajo del estudiante novato. Someter a un alumno de primer año a un test continuo de 10 preguntas complejas de ruteo genera un efecto documentado de fatiga cognitiva, lo que incrementa el riesgo de que el evaluado entregue respuestas azarosas, invalidando así el diagnóstico real de su máquina nocional.

Para mitigar este riesgo, el Instrumento de Diagnóstico Temprano (denominado coloquialmente a lo largo de las intervenciones de aula como cuestionario “Caza Bobos”) se estructuró bajo un **diseño modular**. La batería de 10 reactivos se dividió en dos módulos independientes de 5 preguntas cada uno (Fase 1 y Fase 2), aplicados en momentos distintos del semestre académico, acordes a la progresión del temario. Esta decisión metodológica justifica la existencia de dos tamaños de muestra distintos ($N = 133$ y $N = 103$) durante la fase de recolección de datos.

3.2.2. Batería de Preguntas del Instrumento

En estas preguntas, los distractores (alternativas incorrectas) no fueron generados al azar, sino que fueron diseñados intencionalmente para representar errores conceptuales (*misconceptions*) ampliamente documentados en la literatura científica y empírica de la didáctica de la informática. De esta forma, la alternativa exacta que el estudiante elige permite diagnosticar automáticamente la deficiencia estructural o semántica de su modelo mental.

Cabe destacar que las preguntas expuestas a continuación corresponden a la versión base utilizada para el desarrollo del instrumento. Con el propósito de diversificar la evaluación y analizar la consistencia de las respuestas, se diseñaron variantes para cada reactivo; los formatos alternativos A, B y C para las primeras cinco preguntas, así como los formatos A y B para las cinco preguntas restantes, se encuentran detallados para su revisión en los **Anexos A y B** de este documento⁵.

A continuación, se presenta el cuerpo de preguntas principal que conforma el instrumento diagnóstico, detallando el error que evalúan, la fundamentación teórica de su existencia y el **análisis teórico del distractor**⁶.

1. $x = 10$

⁵El código fuente y la estructura de estas variantes están implementados en el archivo `anexos.tex`.

⁶Se emplea el término “análisis teórico del distractor” dado que el enfoque de este instrumento es cualitativo y formativo, centrado en la validez del constructo del error, prescindiendo así de métricas exclusivas de la psicometría clásica de fiabilidad, como el cálculo del Alfa de Cronbach.

```
y = x
x = 15
print(y)
```

¿Qué valor imprime la consola?

- a) 15
- b) 10
- c) Error

Nombre del error: Variables permanentemente vinculadas (Enlace Dinámico).

Explicación: El estudiante asume que al usar el operador de asignación ($y = x$), ambas variables quedan entrelazadas para siempre, de forma similar al comportamiento de las celdas en una hoja de cálculo. El modelo mental falla al no abstraer que la asignación en lenguajes imperativos evalúa el valor en un instante discreto y realiza una copia desvinculada en la memoria [Weintrop y Wilensky, 2015].

Análisis teórico del distractor: El nivel experto (nivel 3) marca la alternativa correcta (b). El estudiante que acarrea el *misconception* estructural severo (Nivel 1) cae en la trampa (a), creyendo que modificar x a posteriori actualiza automáticamente a y . Quienes marcan (c) suelen presentar una sobrecarga cognitiva o confusión superficial (Nivel 2), asumiendo que reasignar la misma variable genera un error.

```
2. vidas = 3
   vidas = vidas + 1
   print(vidas)
```

¿Qué resultado produce este código?

- a) 4
- b) El programa se detiene con un error matemático.
- c) 3

Nombre del error: Interferencia algebraica en la asignación.

Explicación: Existe una interferencia del conocimiento matemático previo del estudiante con la máquina nocional [Bonar y Soloway, 1985]. Percibe la asignación = como una declaración de igualdad estática matemática (donde $0 = 1$ es una contradicción) en lugar de una "actualización destructiva" secuencial.

Análisis teórico del distractor: El nivel experto (nivel 3) elige (a). El distractor principal es la (b). El estudiante en Nivel 2 (Regular) elige (b) al transferir reglas matemáticas, concluyendo que la expresión representa una paradoja imposible para el computador.

```
3. contador = 0
   if contador < 3:
       print(contador)
       contador += 1
```

¿Qué se imprime en pantalla?

- a) 0 1 2
- b) 0
- c) Error, el if no lleva un incremento adentro.

Nombre del error: Confusión semántica entre Bifurcación e Iteración (if vs while).

Explicación: El estudiante confunde los mecanismos de enrutamiento del código [Sirkiä, 2012]. El modelo mental colapsa al creer que una sentencia condicional (if) repetirá su bloque internamente mientras su condición se mantenga verdadera, atribuyéndole erróneamente capacidades iterativas.

Análisis teórico del distractor: El experto (nivel 3) elige (b). El estudiante en Nivel 1 (Deficiente) cae en el distractor (a), sufriendo una falla estructural pura de enrutamiento. Quien elige (c) está en un Nivel 2, pues aunque entiende cómo fluye la lógica, inventa reglas sintácticas inexistentes producto de la memorización mecánica.

4.

```
intentos = 0
while intentos < 2:
    print("Intentando conexión...")
```

¿Cuántas veces se imprimirá el texto?

- a) 2 veces.
- b) 0 veces.
- c) Infinitas veces.

Nombre del error: Terminación mágica de ciclos (El "Superbug").

Explicación: El estudiante dota a la máquina de inteligencia humana. Asume que el ciclo sabrá cuándo debe detenerse por pura intuición (al ver la condición original), asumiendo que finalizará por sí solo sin necesidad de que se programe una actualización explícita de la variable de control [Pea, 1986].

Análisis teórico del distractor: El nivel experto (nivel 3) elige (c) porque reconoce la falta de un acumulador. El estudiante en Nivel 1 elige (a) al transferir la semántica de la intención humana a la máquina (interferencia). El distractor (b) corresponde al Nivel 2 (Fallo sintáctico o mecánico de ruteo).

5.

```
a = 10
b = 0
c = 0
if a > 0:
    b = 10
else:
    c = 10
print(c)
```

¿Qué valor imprime la variable c?

- a) 10
- b) 0

c) a y b

Nombre del error: Ejecución simultánea de ramas condicionales (if-else).

Explicación: El novato carece del concepto de exclusión mutua de los caminos lógicos. Asume erróneamente que las instrucciones tanto de la rama if como de la rama else se ejecutarán secuencialmente de arriba hacia abajo [Sleeman *et al.*, 1986].

Análisis teórico del distractor: El experto (nivel 3) elige la (b). El distractor caza bobos.^{es} la (a), el cual clasifica al estudiante automáticamente en Nivel 1 (Colapso de Enrutamiento) al evidenciar que su modelo mental evaluó y ejecutó la cláusula `else` a pesar de haber entrado exitosamente al `if`.

6. Desarrollo de Código:

Considere un diccionario que asocia a cada letra su tipo, ya sea una vocal o una consonante. Tanto las llaves como los valores son `strings`.

Escriba la función `obtener_tipo(catalogo, letra)` que reciba como parámetro el diccionario con los tipos de letra y un `string` con una letra. La función debe retornar el tipo de la letra indicada en el segundo parámetro.

Ejemplos:

```
>>> catalogo = {"a": "vocal", "b": "consonante", "e": "vocal"}
>>> print(obtener_tipo(catalogo, "a"))
```

Nombre del error: Dependencia secuencial y uso redundante de ciclos (Falla de acceso $O(1)$).

Explicación: Este comportamiento se documenta bajo los errores de uso redundante o innecesario de estructuras de datos [da Silva, 2024]. El modelo mental del estudiante falla al intentar extrapolar la semántica de una lista (búsqueda secuencial) a un diccionario (que funciona como memoria asociativa mediante un mapeo directo o *hash*). Al no abstraer correctamente la estructura, el novato ignora el acceso directo y obliga a la máquina nocional a realizar una iteración innecesaria.

Análisis teórico del distractor: Al ser una pregunta de desarrollo, la trampa consiste en la inducción a elegir un mal diseño algorítmico. El nivel 3 retorna directamente el valor usando la llave (ej. `return catalogo[nombre]`). El estudiante cae en Nivel 2 (Dependencia Excesiva) si redacta un ciclo `for` para iterar sobre todas las llaves comparando una por una, revelando un colapso en la abstracción al tratar el diccionario como un arreglo lineal.

```
7. lista1 = [1, 2, 3]
   lista2 = lista1
   lista2[0] = 99
   print(lista1[0])
```

¿Qué valor se imprime en consola?

- a) 99
- b) 1
- c) 0

Nombre del error: Creación de copias falsas por asignación de referencias (Aliasing).

Explicación: Los estudiantes novatos presentan dificultades para comprender que existen tipos de datos complejos que no se copian por asignación estática. El estudiante asume erróneamente que la instrucción `lista2 = lista1` crea un clon independiente y desconectado de los datos en la memoria, ignorando que ambas variables pasan a apuntar a la misma referencia física [Ma, 2007].

Análisis teórico del distractor: La alternativa correcta (nivel 3) es 99. El distractor principal es (b) 1. El estudiante elige este distractor si sufre del error de la copia falsa (Nivel 1), creyendo que `lista1` quedó intacta. El distractor secundario (c) detecta a quienes pierden el seguimiento lógico (Nivel 2).

8. `edades = [18, 19, 20]`

`print(edades[2])`

¿Qué se imprime al ejecutarse esta línea?

- a) 2
- b) 20
- c) 19

Nombre del error: Confusión Estructural (Índice vs. Elemento) y Error Off-by-One.

Explicación: Este problema evalúa dos déficits cognitivos en arreglos. El primero es la dificultad para discernir entre la posición de memoria (el índice) y el contenido almacenado en dicha celda [du Boulay, 1986]. El segundo es la transferencia de las reglas de conteo del lenguaje natural (iniciadas en 1) a la máquina nocional, ignorando la indexación base-0.

Análisis teórico del distractor: El experto (nivel 3) elige 20. El distractor más grave es (a) 2, el cual clasifica al estudiante en Nivel 1 (Confusión Estructural Pura) por colapsar el índice con el contenido. El distractor (c) 19 lo ubica en Nivel 2, indicando que entiende la estructura pero padece el error *Off-by-One*.

9. `def configurar_juego():`

`puntos = 100`

`configurar_juego()`

`print(puntos)`

¿Qué se imprime al final?

- a) 100
- b) 0
- c) Da un error (NameError).

Nombre del error: Fusión del Scope Local/Global (Dificultad para discriminar entornos).

Explicación: Los estudiantes carecen del modelo abstracto de la "Pila de Llamadas" (*Call Stack*). Al no visualizar que cada función crea un marco (*frame*) temporal y aislado en memoria, asumen que cualquier variable inicializada dentro de un subprograma se inyecta o hereda permanentemente hacia el espacio global del programa [Fleury, 1991].

Análisis teórico del distractor: El nivel experto (nivel 3) elige (c) comprendiendo la encapsulación modular. El distractor caza bobos.^{es} (a) 100; quien lo elige es ubicado en Nivel 1 por evidenciar la fusión de entornos. El distractor (b) 0 ubica al alumno en Nivel 2 (Fallo de rastreo) al asumir valores globales por defecto ante variables no declaradas.

```
10. def contar_pares(lista_numeros):  
    contador = 0  
    for num in lista_numeros:  
        if num % 2 == 0:  
            contador += 1  
    return contador
```

```
print(contar_pares([-2, 3, 4, 8]))
```

¿Qué valor imprime la consola?

- a) 3
- b) 0
- c) 1

Nombre del error: Retorno prematuro e inconsistente dentro de iteraciones.

Explicación: Los estudiantes ubican la sentencia `return` dentro de ciclos asumiendo equivocadamente su comportamiento temporal [Sirkiä, 2012]. Su modelo mental falla al considerar que la instrucción `return` esperará de manera inteligente a que el ciclo termine actuando como un acumulador, ignorando la regla rígida del computador: el `return` quiebra el flujo secuencial y destruye el entorno en el instante exacto en que se evalúa por primera vez.

Análisis teórico del distractor: El nivel experto (nivel 3) elige (c) 1 (expulsa el valor tras el primer elemento par encontrado). El distractor caza bobos.^{es} (a) 3. El estudiante que lo elige es enviado a Nivel 1 (Retorno Inconsistente) por contar mentalmente los 3 números pares disponibles, asumiendo que el ciclo `for` es inquebrantable a pesar de la indentación del retorno.

3.3. Validación Preliminar de Constructo y Refinamiento

3.3.1. Validación Preliminar de Constructo

Para asegurar la validez teórica del Instrumento de Diagnóstico Temprano, la batería de preguntas en su totalidad fue sometida a un proceso de **Validación Preliminar de Constructo** mediante la revisión exhaustiva del profesor guía de la presente investigación. El resultado de esta validación confirmó que los distractores de los reactivos diseñados poseen una excelente pertinencia para diagnosticar los errores conceptuales documentados en la literatura y evaluar fielmente el estado de la máquina nociónal del estudiante frente a los paradigmas de programación impartidos.

3.3.2. Mitigación de la Fatiga de Prueba y Preguntas Descartadas

Si bien la validación preliminar confirmó la idoneidad teórica de todos los reactivos propuestos inicialmente, se tomó la decisión metodológica de descartar dos ítems de la versión final del instrumento aplicado. El objetivo de esta exclusión fue optimizar el tiempo de resolución en el aula.

Esta decisión se sustenta en la literatura sobre el diseño de instrumentos de evaluación en informática, la cual advierte sobre la necesidad crítica de limitar la extensión de los cuestionarios para evitar los efectos de la “fatiga de prueba” (*testing fatigue*) [Weintrop y Wilensky, 2015]. Evaluaciones excesivamente largas pueden comprometer la concentración del estudiante novato, llevándolo a responder al azar y alterando la fiabilidad del diagnóstico cognitivo real.

A pesar de su exclusión del cuestionario final para mantener la concisión de la prueba, estas dos preguntas poseen un alto valor diagnóstico para detectar deficiencias estructurales en el ámbito de la modularidad y las funciones. Por ello, se documentan a continuación, quedando validadas y disponibles para futuras iteraciones o ampliaciones de la herramienta de diagnóstico en la universidad:

```
1. def calcular_doble(x):  
    print(x * 2)
```

```
resultado = calcular_doble(5)  
suma = resultado + 3  
print(suma)
```

¿Qué ocurre al ejecutar este código?

- a) Se imprime 10 y luego 13.
- b) Da un error matemático (TypeError) porque no se puede sumar a None.

c) Se imprime 13 solamente.

Nombre del error: Confusión semántica entre salidas visuales y retornos de datos (`print` vs `return`).

Explicación: Documentado por [Vásquez Pinto, 2020] en su taxonomía de errores bajo el Misconception ID 28 (“Confusión en el uso de `print` o `return`”), este error se fundamenta en la incomprensión de los mecanismos de comunicación de la máquina nocio-nal. Los estudiantes novatos asumen erróneamente que la instrucción `print` transfiere los datos calculados de vuelta al flujo principal del programa, sin lograr abstraer la diferencia crítica entre una salida por consola (un efecto puramente visual) y el mecanismo formal de traspaso de memoria entre entornos (`return`).

Análisis teórico del distractor: La alternativa correcta es (b), la cual indica el modelo mental experto (nivel 3), puesto que en Python, si una función no posee la sentencia `return`, entrega `None` por defecto, provocando un error lógico al intentar realizar una suma matemática. El distractor principal “caza bobos” es (a), el cual ubica al alumno directamente en un nivel de deficiencia estructural (Nivel 1). El estudiante elige (a) al creer que `print(x * 2)` imprime el 10 e “inyecta” mágicamente ese mismo valor en la variable `resultado`, asumiendo que el flujo luego permitirá sumar $10 + 3 = 13$.

```
2. def sumar_cinco(numero):  
    return numero + 5
```

```
valor = 10  
sumar_cinco(valor)  
print(valor)
```

¿Qué valor imprime la consola al final?

- a) 15
- b) 10
- c) None

Nombre del error: Retorno no almacenado y desconocimiento de la mutación por paso de valor (*Statement with no effect*).

Explicación: Este problema evalúa el error que [da Silva, 2024] cataloga como “MC³ H1: Statement with no effect” y que [Sorva, 2012] denomina “Func6: Failing to store return value”. El modelo mental del estudiante falla al creer que invocar a una función modificará de manera implícita las variables locales entregadas como argumentos. Ignoran el principio de actualización destructiva y pierden por completo el dato retornado al omitir la operación de asignación estática para atrapar el valor.

Análisis teórico del distractor: La alternativa correcta es (b) 10, ya que la función se ejecuta y retorna 15, pero ese valor se pierde en el entorno global al no existir una reasignación; la variable original `valor` permanece inalterada (nivel 3). El distractor “caza bobos” es (a) 15. El estudiante elige este distractor si acarrea el *misconception* (Nivel 1), asumiendo que al simplemente pasar la variable `valor` a la función, el `return` sobrescribe o modifica automáticamente su estado global en

la memoria subyacente sin necesidad de una instrucción explícita como `valor = sumar_cinco(valor)`.

3.4. Diseño de la Rúbrica Formativa Específica

Tras la validación de la batería de preguntas, se procedió a diseñar la Rúbrica Formativa Determinista. A diferencia de las matrices de evaluación tradicionales que agrupan preguntas para calcular un puntaje holístico o calificación final, se optó por un modelo de evaluación granular y atómico por reactivo, alineado a la taxonomía de conocimiento de Qian y Lehman (2017).

3.4.1. Regla de Triage Cognitivo y Criterio de Gravedad Máxima

Dado que el instrumento no busca calificar sino diagnosticar, la Rúbrica Formativa Determinista descarta la promediación lineal de resultados y evalúa el estado cognitivo del estudiante aplicando dos principios fundamentales:

- **Jerarquización del Error (Criterio de Gravedad Máxima):** Cada alternativa de un reactivo está mapeada de manera determinista a uno de 3 niveles de logro del estado de la máquina nocional: **Nivel 1** (*Misconception* o Falla Conceptual Estructural), **Nivel 2** (Falla Superficial, Estratégica o Mecánica) y **Nivel 3** (Logro Esperado / Modelo Mental Correcto). El Criterio de Gravedad Máxima establece que si el estudiante marca el distractor diseñado como *misconception*, es clasificado automáticamente en el nivel más grave (Nivel 1). Para ilustrar esto con un caso numérico concreto: si un alumno responde 4 de las 5 preguntas de la Fase 1 de manera impecable (obteniendo Nivel 3 en P1, P2, P4 y P5), pero en la Pregunta 3 elige el distractor (a) correspondiente a confundir un `if` con un `while` (Nivel 1), su diagnóstico para ese constructo específico de control de flujo se mantiene irrevocablemente en Nivel 1. Sus 4 aciertos no compensan, no promedian, ni borran su falla, ya que acarrea una *Alerta Roja.estructural*. Bajo esta misma regla metodológica, la omisión de respuesta o entrega de código en blanco se categoriza directamente en Nivel 1, asumiendo una parálisis y colapso total de la abstracción.
- **Independencia Conceptual (Triage Cognitivo):** Cada pregunta del Inventario de Conceptos modular opera como una unidad diagnóstica aislada. El Triage Cognitivo permite identificar qué conceptos específicos requieren intervención urgente sin que estos se camuflen entre los aciertos. Al no promediar los resultados, el error crítico se aísla, permitiendo que la acción remedial docente sea exacta y focalizada.

3.5. Matriz Diagnóstica - Fase 1: Fundamentos (Forma A)

En esta primera etapa, el instrumento evalúa los algoritmos básicos de la máquina nomenclativa mediante cinco reactivos centrados exclusivamente en el Modelo de Memoria y el Flujo de Control. En la Tabla 1 se presenta la clasificación determinista de las alternativas para esta fase.

Tabla 1: Rúbrica granular de 3 niveles para la Fase 1. Mapeo correspondiente a la Forma A. Fuente: Elaboración propia basada en la taxonomía de conocimiento de Qian y Lehman (2017).

Reactivo	Nivel 1: Error Conceptual (Misconception Estructural)	Nivel 2: Error Superficial (Interferencia / Sintaxis)	Nivel 3: Logro Esperado (Modelo Mental Correcto)
P1 (Modelo de Memoria)	Elige (a): Cree que la asignación $y = x$ crea un vínculo o enlace permanente e inquebrantable. Falla la abstracción de la memoria.	Elige (c): Falla en el seguimiento de variables múltiples debido a una sobrecarga cognitiva o confusión superficial.	Elige (b): Comprende la actualización destructiva y la temporalidad dinámica de la variable.
P2 (Actualización Destructiva)	Elige (b): Interferencia matemática. Asume que $vidas = vidas + 1$ es una ecuación algebraica irresoluble.	Elige (c): Sobrecarga cognitiva o falta de actualización del valor por arrastre de error.	Elige (a): Domina la semántica de la asignación y la sobrescritura del estado en memoria.
P3 (Condicional vs Ciclo)	Elige (a): Colapso de enrutamiento. Confunde la evaluación única de un <code>if</code> con la cualidad iterativa de un <code>while</code> . Falla estructural de flujo.	Elige (c): Descuido de sintaxis. Entiende la bifurcación, pero inventa reglas (ej. condicional requiere incrementar contadores internos).	Elige (b): Diferencia semántica e intrínsecamente un ciclo inquebrantable de una bifurcación condicional.
P4 (El "Superbug")	Elige (a): Transfiere la semántica de la intención humana a la máquina (interferencia). Asume que el ciclo tiene inteligencia propia para detenerse.	Elige (b): Fallo sintáctico o mecánico de ruteo. Error de rastreo en el límite del ciclo (<i>Off-by-One</i>).	Elige (c): Reconoce la falta de un acumulador y ejecuta el enrutamiento de manera estricta y determinista.
P5 (Ramas Simultáneas)	Elige (a): Colapso de enrutamiento y falla del principio de exclusión mutua. Cree que las ramas de un <code>if-else</code> se pueden ejecutar a la vez.	Elige (c): Fallo de rastreo o confusión sintáctica superficial sobre las instrucciones del bloque.	Elige (b): Comprende el aislamiento, el salto de bloque y la exclusión estricta de las ramas lógicas.

Justificación Teórica de los Distractores (Fase 1) El diseño de los distractores críticos (Nivel 1) en esta matriz no obedece a errores de sintaxis generados al azar, sino a deficiencias cogni-

tivas profundas documentadas en la literatura. Para la dimensión de Modelo de Memoria (P1 y P2), la rúbrica aísla y castiga con la máxima severidad la creencia de los "enlaces permanentes" dinámicos [Weintrop y Wilensky, 2015] y la "interferencia matemática" de la asignación destructiva [Qian y Lehman, 2017], las cuales impiden el rastreo de variables. En la dimensión de Flujo de Control (P3, P4 y P5), los distractores están milimétricamente calibrados para detectar el colapso de enrutamiento (confundir bifurcaciones con ciclos) y el fenómeno del "Superbug" [Sorva, 2012], revelando cuándo un alumno transfiere intenciones humanas a la máquina nocional asumiendo que los ciclos se detienen por intuición.

3.6. Matriz Diagnóstica - Fase 2: Abstracción (Forma A)

El segundo módulo eleva significativamente la complejidad al evaluar constructos de abstracción algorítmica: Estructuras de Datos, Referencias Dinámicas (asignación a objetos mutables) y el *Scope* (alcance) de las funciones. En la Tabla 2 se expone el mapeo de los niveles cognitivos para estos reactivos.

Tabla 2: Rúbrica granular de 3 niveles para la Fase 2. Mapeo correspondiente a la Forma A.
Fuente: Elaboración propia basada en la taxonomía de conocimiento de Qian y Lehman (2017).

Reactivo	Nivel 1: Error Conceptual (Misconception Estructural)	Nivel 2: Error Superficial (Interferencia / Sintaxis)	Nivel 3: Logro Esperado (Modelo Mental Correcto)
P6 (Estructuras: Mapeo)	Código Inválido u Omisión: Colapso para abstraer la estructura. Desconoce por completo cómo operar la estructura de datos avanzada.	Uso de Ciclos (Dependencia): Entiende las colecciones, pero trata a los diccionarios como listas. Ignora la ventaja del mapeo directo.	Acceso Directo por Llave: Comprende la abstracción algorítmica y aprovecha la naturaleza del mapeo <i>hash</i> .
P7 (Memoria: Referencias)	Elige (b): Falsa Copia. Asume que asignar una lista a otra variable crea un clon independiente. Ignora las referencias a objetos mutables.	Elige (c): Falla en rastrear las mutaciones simultáneas de la lista debido a sobrecarga en la memoria de trabajo.	Elige (a): Diferencia el dato del puntero y domina el concepto de referencias dinámicas en memoria.
P8 (Estructuras: Índices)	Elige (a): Confusión estructural pura. Colapsa mentalmente la "posición" de la memoria (el índice) con el "contenido" literal extraído.	Elige (c): Error superficial de posicionamiento (<i>Off-by-One</i>), como asumir que las listas inician en el índice 1 en lugar de 0.	Elige (b): Separa correctamente la arquitectura de direccionamiento de memoria del dato almacenado.
P9 (Scope: Fusión)	Elige (a): Fusión Local/Global. Falla en el encapsulamiento; cree que variables de una función inyectan a la memoria global permanentemente.	Elige (b): Fallo mecánico de traza. Asume que una variable no declarada valdrá 0 por defecto, ignorando el error de sintaxis del intérprete.	Elige (c): Entiende el entorno local (<i>frame</i>), la destrucción de variables y el encapsulamiento.
P10 (Retornos Inconsistentes)	Elige (a): Retorno pasivo. Asume que <code>return</code> es un acumulador que espera a que termine el ciclo. Ignora el quiebre abrupto del flujo.	Elige (b): Error de rastreo matemático de los datos que viajan como parámetros y se retornan.	Elige (c): Comprende que el retorno ejecuta una extracción de datos y un quiebre inmediato en la ejecución de la función.

Justificación Teórica de los Distractores (Fase 2) En esta fase, la rúbrica intercepta el colapso de la abstracción en Python. Para las estructuras de datos (P6 y P8), el Nivel 1 detecta la "confusión estructural pura" (mezclar el índice de memoria con el contenido literal) y castiga con máxima severidad la omisión o código en blanco, considerándola parálisis cognitiva [Sorva, 2012]. A su vez, el Nivel 2 en la P6 aísla estratégicamente a quienes logran funcionalidad pero presentan una "dependencia de ciclos", ignorando las ventajas del mapeo *hash*. En cuanto a la modularidad (P9 y P10), los distractores de Nivel 1 revelan la "fusión

local/global”(falta de encapsulamiento temporal) y el retorno pasivo”, evidenciando que el novato no logra abstraer el quiebre abrupto de flujo que ejecuta internamente la instrucción `return` [da Silva, 2024]. Por último, la P7 se alinea con la literatura moderna al aislar el *misconception* de la “falsa copia”, diferenciando tajantemente a los alumnos que no han asimilado la asignación de objetos mutables propia de la semántica de Python.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

El presente capítulo expone los resultados empíricos obtenidos tras la aplicación del Inventario de Conceptos y la ejecución de la Rúbrica Diagnóstica. Se detalla el escenario logístico en el que se efectuaron las mediciones y se presenta el diagnóstico cuantitativo del estado de la “máquina nocional” de los estudiantes, dividido en las dos fases de aplicación modular del instrumento. Finalmente, se exploran las implicancias pedagógicas mediante análisis visuales y un estudio de caso docente, declarando las limitaciones inherentes al diseño empírico.

4.1. Descripción del Escenario de Validación

Para llevar a cabo la recolección de datos de la propuesta, la medición se integró directamente en la dinámica regular del curso de Programación. Con el objetivo de obtener una muestra diversa y evitar sesgos demográficos locales, el instrumento diagnóstico fue aplicado a estudiantes de distintos cursos paralelos pertenecientes tanto al Campus Santiago (San Joaquín) como a la Casa Central (Valparaíso) de la Universidad Técnica Federico Santa María.

Las mediciones se realizaron de forma presencial dentro del aula de clases durante los bloques horarios de la asignatura, garantizando así un entorno controlado con supervisión docente, mitigando la alteración de datos que suele ocurrir en evaluaciones asincrónicas no supervisadas. Siguiendo el diseño modular del instrumento, la aplicación se dividió en dos hitos evaluativos:

- **Aplicación de la Fase 1 (Preguntas 1 a la 5):** Este primer módulo evaluativo se aplicó durante las clases correspondientes a los días miércoles. El contexto temporal de esta fase estuvo marcado por la proximidad de las evaluaciones sumativas oficiales del curso, lo cual proporcionó un escenario donde los estudiantes exhibían un nivel de preparación académica considerable. La muestra válida recolectada en esta fase fue de $N = 133$ estudiantes.
- **Aplicación de la Fase 2 (Preguntas 6 a la 10):** Debido a circunstancias logísticas externas, la aplicación de este segundo módulo debió ser reprogramada de un día miércoles a un día lunes. Este cambio de horario generó un aumento en la inasistencia estudiantil, provocando una caída en la muestra válida de $N = 133$ a $N = 103$ estudiantes. Esta variación constituye un posible sesgo de selección que debe ser considerado al interpretar los datos inter-fases. Adicionalmente, el inicio de la semana académica implicaba que el dominio de ciertas materias en esta fase dependía intrínsecamente del estudio autónomo del alumno (“cápsulas de preclase”) durante el fin de semana.

4.2. Resultados de la Fase 1: Fundamentos de Programación

4.2.1. Tratamiento de Respuestas Omitidas

Durante el proceso de recolección de datos, se observó que algunos estudiantes dejaron reactivos en blanco. En el contexto de este diagnóstico temprano, una respuesta omitida frente a un reactivo de código elemental sugiere una incapacidad basal para iniciar el ruteo. Por consiguiente, la rúbrica aplicó de manera estricta la Regla de Precedencia Crítica, clasificando a estas omisiones directamente en el Nivel 1: Deficiente (Misconception Estructural).

4.2.2. Análisis Cuantitativo del Diagnóstico

Una vez homologados los datos provenientes de las distintas formas del instrumento (A, B y C) hacia la Forma A, la matriz procedió a clasificar a los 133 estudiantes en los diferentes niveles de logro. Los datos absolutos y porcentuales de las respuestas se detallan en la Tabla 3.

Tabla 3: Frecuencia y clasificación cognitiva de las respuestas obtenidas en la Fase 1 ($N = 133$). Los datos han sido homologados a la Forma A del instrumento.

Fuente: Elaboración propia.

Reactivo	Dimensión a Evaluar	Nivel 3 (Correcto)	Nivel 1 (Crítico)	Nivel 2 (Superficial)	Omisiones (Nivel 1)
P1	Modelo de Memoria (Enlaces Permanentes)	Alternativa (b)102 (76,7 %)	Alternativa (a)29 (21,8 %)	Alternativa (c)2 (1,5 %)	0 (0,0 %)
P2	Modelo de Memoria (Interferencia Matemática)	Alternativa (a)130 (97,7 %)	Alternativa (b)3 (2,3 %)	Alternativa (c)0 (0,0 %)	0 (0,0 %)
P3	Flujo de Control (if vs while)	Alternativa (b)66 (49,6 %)	Alternativa (a)54 (40,6 %)	Alternativa (c)13 (9,8 %)	0 (0,0 %)
P4	Flujo de Control (El "Superbug")	Alternativa (c)111 (83,5 %)	Alternativa (a)22 (16,5 %)	Alternativa (b)0 (0,0 %)	0 (0,0 %)
P5	Flujo de Control (Ejecución Simultánea)	Alternativa (b)117 (88,0 %)	Alternativa (a)16 (12,0 %)	Alternativa (c)0 (0,0 %)	0 (0,0 %)

4.2.3. Discusión de los Resultados y Observaciones Teóricas

La separación de los datos objetivos de su respectiva interpretación teórica permite extraer observaciones consistentes con la literatura en educación informática:

1. **Estabilidad en la actualización destructiva:** Los datos sugieren una alta asimilación de la actualización destructiva básica (P2), donde el 97,7 % de la muestra superó la posible interferencia del álgebra tradicional ($vidas = vidas + 1$). Esto sugiere que la interferencia matemática no representó un problema generalizado en esta cohorte.

2. **Prevalencia de la noción de “enlace permanente”:** En contraste, la P1 revela que un 21,8 % de los alumnos asume que la asignación $y = x$ crea un lazo temporal inquebrantable. Este comportamiento proporciona evidencia a favor de lo documentado por Weintrop y Wilensky (2015) sobre los *misconceptions* de variables ligadas permanentemente, sugiriendo deficiencias en la comprensión de la temporalidad dinámica de la memoria.
3. **Severidad en la abstracción del flujo de control:** El área que reporta mayor vulnerabilidad es el enrutamiento lógico. En la P3, un 40,6 % de la muestra fue catalogada en Nivel 1 al confundir la semántica condicional (*if*) con la iteración (*while*). Estos hallazgos son consistentes con la literatura que sitúa al control de flujo como una de las barreras de abstracción más pronunciadas para los programadores novatos.

4.3. Resultados de la Fase 2: Abstracción y Modularidad

4.3.1. Análisis Cuantitativo del Diagnóstico

La segunda fase interceptó el estado de la máquina nocional frente a las Estructuras de Datos, las Funciones y el *Scope*. Los resultados cuantitativos procesados por la Rúbrica Formativa Determinista se detallan en la Tabla 4.

Tabla 4: Frecuencia y clasificación cognitiva de las respuestas obtenidas en la Fase 2 ($N = 103$). Los datos han sido homologados a la Forma A del instrumento.

Fuente: Elaboración propia.

Reactivo	Dimensión a Evaluar	Nivel 3 (Correcto)	Nivel 1 (Crítico)	Nivel 2 (Superficial)	Omisiones (Nivel 1)
P6	Estructuras de Datos (Mapeo y Diccionarios)	Desarrollo 41 (39,8 %)	Código Inválido 36 (35,0 %)	Uso de Ciclos 26 (25,2 %)	0 (0,0 %)
P7	Memoria Dinámica (Referencias en Listas)	Alternativa (a) 52 (50,5 %)	Alternativa (b) 50 (48,5 %)	Alternativa (c) 1 (1,0 %)	0 (0,0 %)
P8	Estructuras de Datos (Índices vs Contenido)	Alternativa (b) 87 (84,5 %)	Alternativa (a) 13 (12,6 %)	Alternativa (c) 3 (2,9 %)	0 (0,0 %)
P9	Scope de Funciones (Inyección Global)	Alternativa (c) 80 (77,7 %)	Alternativa (a) 18 (17,5 %)	Alternativa (b) 5 (4,9 %)	0 (0,0 %)
P10	Flujo en Funciones (return en iteraciones)	Alternativa (c) 11 (10,7 %)	Alternativa (a) 85 (82,5 %)	Alternativa (b) 7 (6,8 %)	0 (0,0 %)

4.3.2. Discusión de los Resultados y Observaciones Teóricas

El análisis de la Tabla 4 sugiere la coexistencia de modelos mentales en desarrollo con vacíos conceptuales considerables:

1. **Variable de confusión en Estructuras (P6):** Aunque la P6 reporta un fracaso estructural del 35,0 % al evaluar el manejo de diccionarios (código inválido + omisiones), es

imperativo reconocer que este resultado podría estar fuertemente influenciado por una variable de confusión: la falta de exposición al material pre-clase (cápsulas asincrónicas). Por tanto, este porcentaje podría reflejar una ausencia de estudio previo del alumno más que un colapso conceptual genuino de la máquina nocional. Por otro lado, un 25,2 % (Nivel 2) forzó el uso de ciclos en lugar del acceso directo por llave, lo que sugiere una dependencia estratégica hacia las listas y una falta de asimilación de la abstracción de mapeo *hash*.

2. **Dificultad con la asignación de objetos mutables (P7):** En la P7, el 44,7 % asume que la asignación de una lista a otra variable crea un “clon” independiente. Esto sugiere que casi la mitad de los estudiantes arrastra el *misconception* de la “falsa copia”, ignorando el modelo de referencias a objetos mutables (o paso por asignación de objeto) propio de la semántica de Python.
3. **Retornos Incomprendidos (P10):** El 78,6 % de los alumnos fue catalogado en Nivel 1 Crítico en P10 al asumir que la sentencia `return` dentro de un ciclo `while` actuaría como un acumulador que “espera” a que el bucle termine. Esto apoya la hipótesis de que los estudiantes ignoran el quiebre abrupto de flujo que produce la extracción de datos en funciones.

4.4. Evaluación del Potencial Diagnóstico para la Gestión Docente

Tal como advierte Silva (2024) en su análisis a gran escala sobre los Problemas de Comprensión en Códigos Correctos (MC³), la identificación y catalogación automatizada de los errores conceptuales no solo beneficia al estudiante, sino que constituye una herramienta de inmenso valor para los instructores. Al revelar las deficiencias estructurales que tradicionalmente quedan ocultas, los docentes pueden tomar conciencia del estado cognitivo real de su clase y adaptar sus estrategias pedagógicas para mitigar la ocurrencia de estos errores.

4.4.1. Generación del Panel Analítico (Dashboard Docente)

Se consolidaron los resultados de los 10 reactivos del Inventario de Conceptos. Los datos fueron agrupados estrictamente bajo los 3 niveles de conocimiento de la taxonomía de Qian y Lehman (2017). Este formato de visualización omite intencionalmente las alternativas específicas marcadas por los alumnos y le presenta al docente directamente la clasificación clínica de la severidad del error (Tabla 5).

Tabla 5: Dashboard analítico consolidado para el docente. Fase 1 ($N = 133$) y Fase 2 ($N = 103$).

Fuente: Elaboración propia a partir de la intercepción de la Rúbrica Determinista.

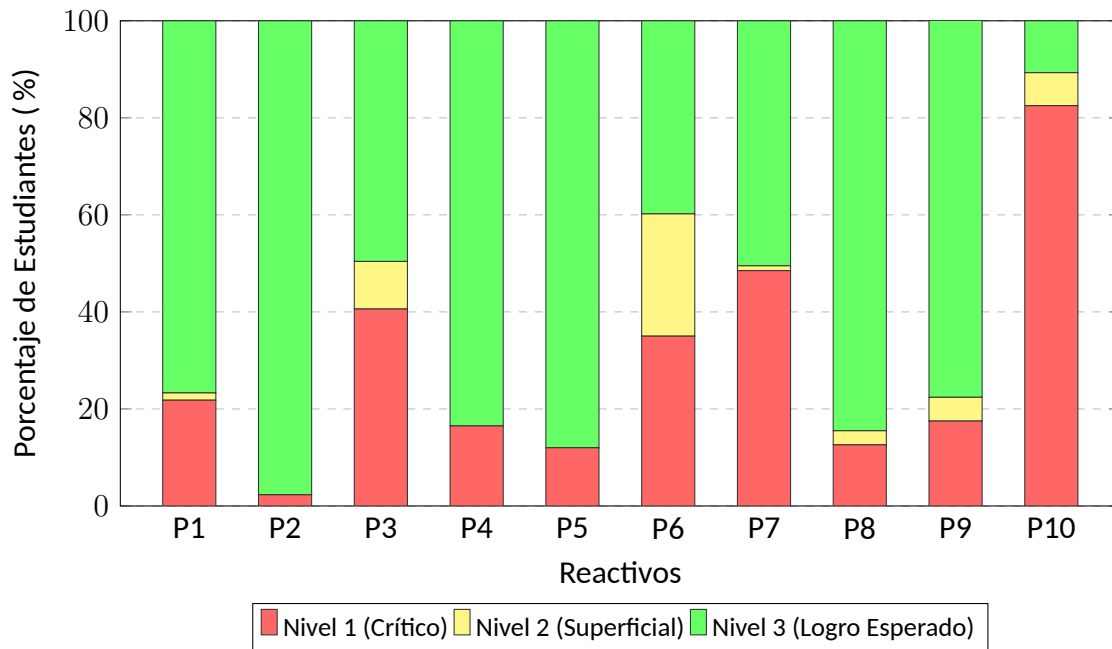
Pregunta	Concepto Evaluado	Nivel 1: Error Conceptual(Crítico)	Nivel 2: Error Superficial(Estratégico / Sintaxis)	Nivel 3: Logro Esperado(Correcto)
P1	Modelo de Memoria: Asignación y enlaces temporales.	29 (21,8 %)	2 (1,5 %)	102 (76,7 %)
P2	Actualización Destructiva e interferencia algebraica.	3 (2,3 %)	0 (0,0 %)	130 (97,7 %)
P3	Flujo de Control: Condicional vs. Iteración (if vs while).	54 (40,6 %)	13 (9,8 %)	66 (49,6 %)
P4	Intuición de la Máquina: Terminación mágica.	22 (16,5 %)	0 (0,0 %)	111 (83,5 %)
P5	Exclusión Mutua: Ejecución de ramas simultáneas.	16 (12,0 %)	0 (0,0 %)	117 (88,0 %)
<i>Transición a la Fase 2 de Abstracción</i>				
P6	Diccionarios: Búsqueda y acceso por llave.	36 (35,0 %)	26 (25,2 %)	41 (39,8 %)
P7	Referencias en Memoria: Paso por valor vs. referencia en listas.	50 (48,5 %)	1 (1,0 %)	52 (50,5 %)
P8	Listas: Diferenciación entre Índice y Contenido.	13 (12,6 %)	3 (2,9 %)	87 (84,5 %)
P9	Scope: Inyección global de variables locales.	18 (17,5 %)	5 (4,9 %)	80 (77,7 %)
P10	Funciones: Comportamiento del return dentro de un ciclo.	85 (82,5 %)	7 (6,8 %)	11 (10,7 %)

Análisis Preliminar y Visualización del Estado Cognitivo Para maximizar la utilidad de esta información como insumo de analítica de aprendizaje (*Learning Analytics*), los datos tabulados se tradujeron a representaciones visuales. Estas gráficas permiten al docente identificar de manera instantánea y clínica los déficits estructurales de la cohorte para su oportuna intervención.

En la Figura 2 se presenta el panorama general mediante un gráfico de barras apiladas, evidenciando visualmente la evolución del curso y la alarmante prevalencia de errores estructurales (Nivel 1, sector inferior de las barras) en reactivos específicos asociados a las dimensiones de control de flujo y abstracción modular.

Figura 2: Distribución porcentual de los modelos mentales por reactivo. Se observa la prevalencia de errores estructurales (Nivel 1) en las preguntas P3 y P10.

Fuente: Elaboración propia.



A partir de la visión consolidada y las gráficas detalladas extraídas del sistema, la herramienta permite detectar al menos cuatro patrones críticos que sirven como justificativo empírico para la replanificación docente:

1. **Colapso Severo en Abstracción y Enrutamiento (Alertas Urgentes):** El reporte evidencia visualmente que los mayores quiebres de la máquina nociónal ocurren en las dimensiones de control de flujo y modularidad. Como se aprecia en la Figura 3, la Pregunta 10 representa el colapso cognitivo más severo de todo el instrumento: un 82,5 % de la clase comete un error crítico al asumir que la instrucción `return` opera como un acumulador cíclico. Del mismo modo, en la fase de fundamentos, la Figura 4 destaca un 40,6 % de “colapso de enrutamiento”, donde los alumnos fallan en distinguir la semántica de una bifurcación (`if`) frente a una iteración (`while`).

Figura 3: Distribución de modelos mentales en la Pregunta 10 (`return` en ciclos). Representa el colapso cognitivo más severo del instrumento (82,5 % de error crítico).

Fuente: Elaboración propia.

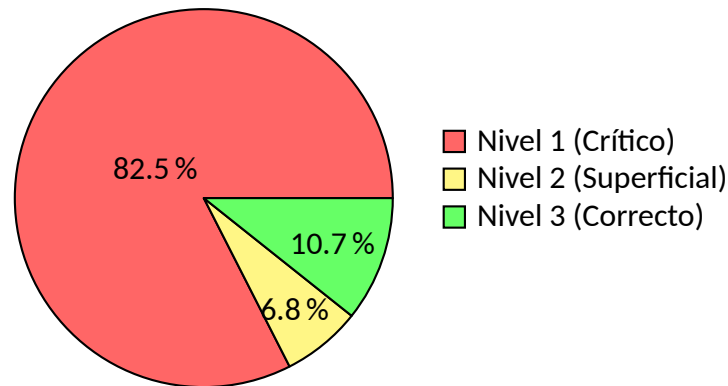
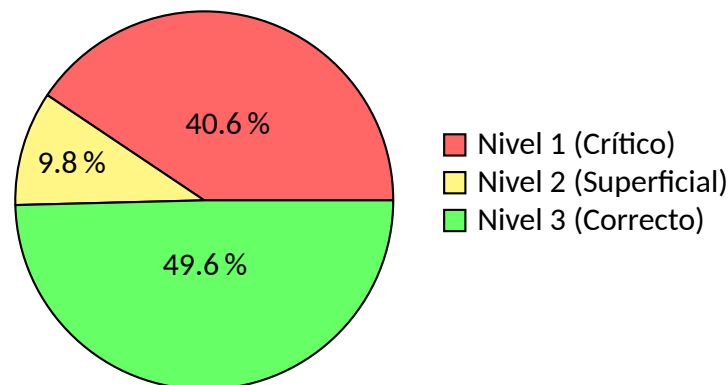


Figura 4: Distribución de modelos mentales en la Pregunta 3 (`if` vs `while`). Destaca un 40,6 % de colapso de enrutamiento.

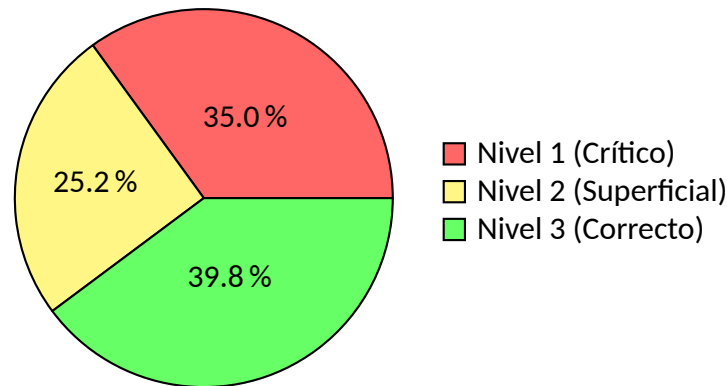
Fuente: Elaboración propia.



2. **Fallas Estratégicas vs. Fallas Conceptuales (Lectura del Nivel 2):** A diferencia de las plataformas de corrección tradicionales que invisibilizan los procesos intermedios, esta matriz revela importantes matices de aprendizaje. La Figura 5 ilustra el caso de la Pregunta 6 (Manejo de Diccionarios). Se observa una fuerte fragmentación y dependencia de ciclos: aunque existe un grueso de error conceptual crítico, el gráfico aísla un bloque categorizado en Nivel 2. Esto le informa al docente que un sector considerable del curso sí logra idear códigos funcionales, pero sufren de una “dependencia estratégica” a las listas, forzando ciclos (`for`) e ignorando completamente las ventajas del mapeo `hash`.

Figura 5: Distribución de modelos mentales en la Pregunta 6 (Diccionarios). Se observa una fuerte fragmentación y dependencia de ciclos (Nivel 2).

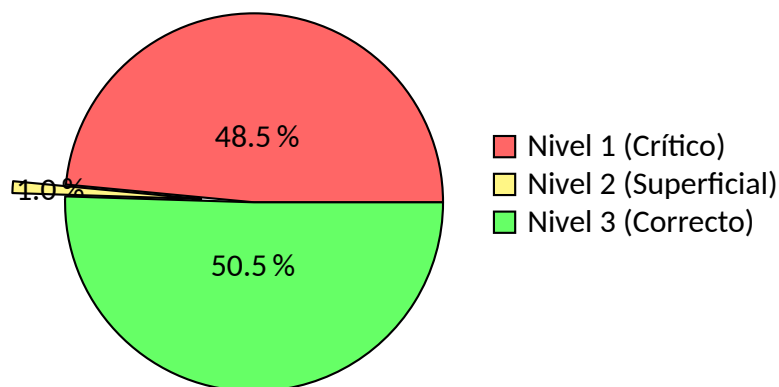
Fuente: Elaboración propia.



3. **Polarización del Modelo de Memoria Dinámica:** Al observar la Pregunta 7 (Clon vs Referencia en Listas), la herramienta revela un fenómeno psicométrico particular. Tal como ilustra la Figura 6, existe una polarización casi simétrica entre el dominio y la falla estructural. El concepto de “paso por asignación de objeto” derechamente divide a la cohorte en dos extremos: aquellos que lograron asimilar el dinamismo de las referencias en memoria y aquellos que siguen arrastrando el *misconception* de los enlaces estáticos y la falsa copia.

Figura 6: Distribución de modelos mentales en la Pregunta 7 (Referencias en listas). Evidencia una polarización casi simétrica entre dominio y falla estructural.

Fuente: Elaboración propia.



4. **Validación de Logros Pedagógicos (Alta prevalencia en Nivel 3):** El dashboard no solo detecta falencias, sino que también retroalimenta positivamente el éxito de la intervención docente. Observando nuevamente la Figura 2, reactivos como P2 y P8 exhiben barras superiores de dominio absoluto (97,7 % y 84,5 % en Nivel 3, respectivamente).

Esta métrica le asegura al profesor que fenómenos como la “interferencia matemática” en asignaciones destructivas han sido superados.

4.5. Sondeo Exploratorio de Percepción Docente

Para explorar empíricamente la recepción de este Panel Analítico, los resultados visuales consolidados fueron presentados formalmente a dos profesores titulares de la asignatura ($N = 2$). Se aplicó un cuestionario cualitativo de percepción docente estructurado en la utilidad del diagnóstico, la toma de decisiones y la proyección futura.

Al tratarse de una muestra reducida, los resultados de este sondeo no pretenden establecer una validación institucional generalizable, sino proporcionar un estudio exploratorio sobre el valor formativo de la herramienta en el aula. A continuación, se destacan las impresiones, convergencias y limitaciones reportadas por los docentes encuestados:

Percepción del Diagnóstico Ambos docentes valoraron la capacidad de la herramienta para realizar un triaje cognitivo inicial. El primer docente indicó que este formato *“permite enfocarse en las falencias críticas, pero a la vez sin dejar de lado las alertas por otras que no lo son tanto”*. El segundo docente concordó, añadiendo que hacer la distinción de niveles de gravedad es *“muy importante porque permite definir cuál es la mejor estrategia para abordar el error”*.

Frente a los hallazgos revelados, los profesores mostraron sorpresa ante la persistencia de ciertos errores, pero también confirmaron la precisión del instrumento respecto a sus propias observaciones en el aula. El segundo docente validó el colapso detectado en la Pregunta 6 (Diccionarios):

“Se alinea totalmente a lo que vi ahora corrigiendo los recuperativos, especialmente eso de que usan el `for` para acceder a las llaves del diccionario [...] Me alivia saber que el `i = i+1` está dominado y no mecanizado solamente, porque lo usamos muchísimo.”

Por su parte, el hallazgo más crítico del estudio (82,5 % de falla en el `return`) generó reacciones complementarias. Mientras el primer docente admitió que *“no lo percibía tan grave”*, el segundo docente sugirió una hipótesis metodológica para explicar este déficit estructural: *“La P10 no la he visto tanto, justamente porque no la hemos ensayado mucho y puede ser que por eso sea tan alto el nivel crítico.”*

Toma de Decisiones y Replanificación Pedagógica Al indagar sobre las estrategias remediales, ambos profesores convergieron unánimemente en las zonas de intervención urgente:

diccionarios y retornos de funciones. El segundo docente señaló categóricamente: *“Los diccionarios!!! Creo que el reporte muestra que no los están entendiendo. Y también el `return` dentro del ciclo, que me permite repasar funciones y ciclos.”*

Para reparar estos colapsos estructurales, las metodologías propuestas por los instructores se alinean directamente con la literatura en didáctica informática. Mientras que para la abstracción de memoria recomendaron representaciones visuales (*“intentar algo gráfico para las estructuras de datos”*), para el control de flujo de ciclos y funciones ambos docentes concluyeron que la simulación paso a paso es la solución idónea (*“además del ruteo para el caso de los ciclos y `return`”*). Esto proporciona evidencia práctica que apoya las propuestas de ruteo guiado expuestas por Sorva (2012) [Sorva, 2012].

Al evaluar la viabilidad de incorporar este instrumento en la gestión académica masiva, la recepción fue extraordinariamente favorable. Frente a la consulta sobre la integración permanente de la herramienta, las respuestas de ambos docentes sugieren un alto potencial de adopción institucional. El segundo docente, proyectando la integración tecnológica del instrumento hacia los Sistemas de Gestión de Aprendizaje (LMS) de la universidad, solicitó:

“Definitivamente! Quisiera tener este tipo de preguntas como cuestionario semanal, y poder acceder al dashboard desde Aula⁷ para poder preparar las clases. Por favor, hazlo un paquete LTI⁸ y lo sumamos para el otro semestre.”

A pesar de este entusiasmo, el sondeo exploratorio también permitió revelar deficiencias de usabilidad en la presentación de los datos. El segundo profesor advirtió que la visualización aislada del panel puede generar confusión si no se acompaña del contexto: *“No me acuerdo cuáles eran exactamente las preguntas, entonces no supe distinguir fácilmente cuáles eran de sintaxis y cuáles eran más profundas. Tampoco me queda claro cuál es fase 1 y fase 2.”* Esto sugiere una limitación en la entrega de la información: el Panel Analítico requiere estar estrictamente vinculado al enunciado del reactivo original para que el instructor no pierda el contexto de la evaluación.

4.6. Limitaciones del Estudio

Para contextualizar adecuadamente los hallazgos de esta investigación, es necesario declarar las siguientes limitaciones metodológicas:

⁷Aula es la plataforma virtual de aprendizaje (LMS) institucional de la Universidad Técnica Federico Santa María, construida sobre la arquitectura de Moodle.

⁸LTI (*Learning Tools Interoperability*) es un estándar global desarrollado por el 1EdTech Consortium que permite integrar de manera estandarizada y segura herramientas de educación de terceros directamente en un entorno LMS.

1. **Sesgo de selección por caída de muestra inter-fases:** La variación temporal del instrumento (aplicación de la Fase 2 un día lunes en lugar del día miércoles habitual) generó una reducción en la muestra de $N = 133$ a $N = 103$ estudiantes. Esta atrición involuntaria impide un seguimiento correlacional directo uno-a-uno entre los alumnos de ambas fases y constituye un posible sesgo de selección a considerar al proyectar las tasas de error.
2. **Presencia de variables de confusión:** Tal como se discutió en los resultados de la Pregunta 6 (Diccionarios), existe una variable de confusión ligada al modelo de aula invertida de la institución. Las tasas de fracaso estructural en constructos avanzados podrían estar infladas por la falta de visualización del material pre-clase asincrónico por parte de los alumnos, más que por una incapacidad cognitiva genuina. Esta misma limitación fue advertida en la P10 por el equipo docente, quienes indicaron que la falta de ensayo previo en ayudantías podría haber maximizado la tasa de error.
3. **Tamaño de la muestra docente:** El *Sondeo Exploratorio de Percepción Docente* se fundamenta en la retroalimentación de dos profesores titulares ($N = 2$). Si bien la convergencia en sus respuestas ofrece una visión rica cualitativamente y reduce el sesgo respecto a un solo individuo, esta muestra sigue siendo limitada e impide extrapolar estadísticamente la aceptación institucional a la totalidad de los profesores del departamento.
4. **Contexto visual del reporte:** El estudio reveló que la entrega del *Panel Analítico* sin los enunciados explícitos de los reactivos dificulta la lectura autónoma por parte del profesorado, constituyendo una limitación de usabilidad de la herramienta que requiere mejoras de interfaz gráfica.
5. **Ausencia de métricas de confiabilidad formales:** La rúbrica y los reactivos diseñados operaron bajo un modelo de validación de constructo preliminar y formativo. Este estudio carece de cálculos psicométricos clásicos de confiabilidad interna (tales como el coeficiente Alfa de Cronbach o de Kuder-Richardson), lo que limita la certificación del instrumento bajo estándares rigurosos de estandarización psicométrica a gran escala.

CAPÍTULO 5

CONCLUSIONES

El aprendizaje de la programación en cursos introductorios de ingeniería representa un desafío cognitivo significativo, donde los estudiantes novatos frecuentemente desarrollan *misconceptions* (errores conceptuales profundos) que bloquean la comprensión de la lógica computacional. Dado que los sistemas de evaluación tradicionales, de carácter sumativo y punitivo, tienden a enmascarar estas deficiencias estructurales al calificar únicamente la correctitud sintáctica del código final, la presente investigación propuso explorar un cambio de paradigma hacia el diagnóstico formativo.

Para abordar esta problemática, en este trabajo se diseñó e implementó un Inventario de Conceptos modular, acoplado a una Rúbrica Formativa Determinista de 3 niveles de logro. A continuación, se detallan las principales conclusiones de este trabajo, estructuradas en estricta alineación con los objetivos específicos planteados.

5.1. Identificación y categorización de *misconceptions*

En relación con el primer objetivo, orientado a fundamentar y operacionalizar las categorías de errores más frecuentes, la investigación logró establecer una conexión directa entre la teoría de la didáctica informática y la práctica evaluativa. Se definieron 4 familias de *misconceptions* aterrizadas a la semántica del lenguaje Python: Modelo de Memoria, Flujo de Control, Estructuras de Datos y *Scope* de Funciones.

Al operacionalizar estas 4 familias, el análisis teórico es consistente con la idea de que errores como la "falsa copia" de listas o el "Superbug" (transferir intuición humana a la máquina) no son simples descuidos de sintaxis, sino manifestaciones de modelos mentales inviables. El diseño de los distractores aporta indicios de que es posible formular alternativas incorrectas que actúen como "atrapadores" de deficiencias cognitivas específicas, sugiriendo un estándar metodológico para la creación de futuras evaluaciones institucionales.

5.2. Diseño y resultados empíricos de la rúbrica determinista

Respecto al diseño del instrumento y la Rúbrica Formativa Determinista, los resultados empíricos sugieren una alta precisión diagnóstica por parte de la herramienta. La aplicación de la matriz granular de 3 niveles de logro permitió visibilizar los quiebres estructurales ocultos en la cohorte.

Los datos obtenidos proporcionan evidencia de fallas críticas de gran magnitud: un 82,5 % de

los estudiantes evidenció un colapso severo al ignorar el quiebre de flujo de un `return` dentro de un ciclo, y un 40,6 % exhibió un colapso de enrutamiento al confundir la bifurcación de un `if` con la iteración de un `while`. Simultáneamente, el instrumento reflejó asimilación en conceptos como la actualización destructiva (97,7 %) y el uso de índices de memoria (84,5 %). La aplicación del Criterio de Gravedad Máxima demostró ser consistente teóricamente, ya que impidió la promediación lineal de notas y aisló exitosamente a los alumnos en riesgo inminente.

5.3. Impacto del diagnóstico en la gestión docente

Al evaluar la utilidad de la Rúbrica Formativa Determinista como sistema de alerta temprana, el impacto exploratorio observado es altamente favorable. La consolidación de los datos en un Panel Analítico sugiere que la herramienta es capaz de guiar la toma de decisiones del equipo docente antes de las evaluaciones sumativas.

El Sondeo Exploratorio de Percepción Docente evidenció la utilidad práctica del instrumento: los profesores a cargo manifestaron sorpresa ante el volumen de fallas en conceptos que habían sido enseñados explícitamente en clases, lo cual es consistente con la resiliencia teórica que caracteriza a los *misconceptions*. La información entregada permitió a los docentes planificar acciones remediales focalizadas, como el ruteo manual guiado. El interés explícito de los profesores encuestados por integrar este diagnóstico como cuestionario recurrente, sugiere que la propuesta metodológica constituye una solución factible y demandada por la gestión académica.

5.4. Limitaciones del estudio

A pesar de los hallazgos prometedores, esta investigación está sujeta a diversas limitaciones metodológicas que deben considerarse al interpretar los resultados. En primer lugar, el estudio se aplicó exclusivamente en una institución, lo que sugiere cautela al intentar generalizar las frecuencias de errores observadas a otras poblaciones estudiantiles. En segundo lugar, se registró una caída en la muestra entre la Fase 1 ($N = 133$) y la Fase 2 ($N = 103$) producto de reprogramaciones horarias, lo que podría introducir un sesgo de atrición al analizar los datos inter-fases. En tercer lugar, el Sondeo Exploratorio de Percepción Docente contó con la participación de dos profesores ($N = 2$); si bien ofrece valor cualitativo, este tamaño muestral impide extrapolar conclusiones sobre la adopción a todo el departamento académico. Finalmente, el alcance del estudio no contempla una evaluación de impacto longitudinal que permita asegurar si las alertas tempranas mejoraron el rendimiento final de los alumnos.

5.5. Trabajo futuro

A partir de la evidencia recopilada, se han identificado las siguientes áreas de oportunidad para futuras investigaciones orientadas a potenciar el diagnóstico temprano:

5.6. Ampliación a nuevas unidades temáticas

El instrumento modular actual cubre los fundamentos de programación procedural evaluados en las primeras etapas del curso. Como trabajo futuro, se propone diseñar nuevos reactivos orientados a la Programación Orientada a Objetos (POO), mapeando *misconceptions* asociados a la instanciación de clases, herencia y paso de mensajes.

5.7. Integración en plataformas de evaluación continua

Dado que el Sondeo Exploratorio evidenció el interés docente en automatizar este proceso, se sugiere integrar la Rúbrica Formativa Determinista como un Paquete LTI dentro de los Sistemas de Gestión de Aprendizaje (LMS) de la universidad, permitiendo que el Panel Analítico se actualice de manera nativa tras cada sesión.

5.8. Medición del impacto longitudinal en los modelos mentales

Para corroborar la eficacia real de las acciones remediales guiadas por este instrumento, se hace necesario un estudio longitudinal. En lugar de evaluar el éxito futuro utilizando únicamente las calificaciones de los certámenes sumativos, el enfoque debe centrarse en medir la "supervivencia de los modelos mentales viables.^a largo plazo. Es decir, evaluar si los estudiantes que recibieron mitigación temprana logran transferir estas estructuras cognitivas correctas hacia asignaturas de programación de ciclos superiores sin recaer en los *misconceptions* basales.

En conclusión, la síntesis de los hallazgos aporta indicios de que la Rúbrica Formativa Determinista es una alternativa metodológica prometedora basada en evidencia, que complementa de forma robusta la evaluación sumativa tradicional, aunque permanece sujeta a las limitaciones propias de un primer despliegue exploratorio.

ANEXOS

En esta sección se adjunta el material complementario que respalda la validez empírica de la investigación. Se presentan los instrumentos de evaluación diseñados para capturar los errores conceptuales en estudiantes de primer año.

Anexo A: Instrumento de Diagnóstico - Iteración 1 (Estructuras de Control)

Este anexo contiene el código fuente de los cuestionarios aplicados a la muestra de 133 estudiantes. El instrumento fue diseñado en tres versiones (Formas A, B y C) para mitigar la copia entre pares, manteniendo la equivalencia en la evaluación de los modelos mentales.

FORMA A - Cuestionario de Evaluación Diagnóstica

1. `x = 10`
`y = x`
`x = 15`
`print(y)`

¿Qué valor imprime la consola?

- a) 15
- b) 10
- c) Error

2. `vidas = 3`
`vidas = vidas + 1`
`print(vidas)`

¿Qué resultado produce este código?

- a) 4
- b) Error matemático.
- c) 3

3. `contador = 0`
`if contador < 3:`
 `print(contador)`
 `contador += 1`

¿Qué se imprime en pantalla?

- a) 0 1 2
- b) 0
- c) Error de incremento.

4. `intentos = 0`
`while intentos < 2:`
 `print("Intentando conexión...")`

¿Cuántas veces se imprimirá el texto?

- a) 2 veces.
- b) 0 veces.
- c) Infinitas.

5. `a = 10`
`b = 0`
`c = 0`
`if a > 0:`
 `b = 10`
`else:`
 `c = 10`
`print(c)`

¿Qué valor imprime la variable c?

- a) 10
- b) 0
- c) a y b

FORMA B - Cuestionario de Evaluación Diagnóstica

1. `a = 5`
`b = a`
`a = 20`
`print(b)`

¿Qué valor imprime la consola?

- a) 5
- b) Error
- c) 20

2. `puntos = 5`
`puntos = puntos + 2`
`print(puntos)`

¿Qué resultado produce este código?

- a) 5
- b) 7
- c) Error matemático.

3. `i = 1`
`if i < 4:`
 `print(i)`
 `i += 1`

¿Qué se imprime en pantalla?

- a) Error de incremento.
- b) 1 2 3
- c) 1

4. `cargas = 0`
 `while cargas < 3:`
 `print("Cargando...")`

¿Cuántas veces se imprimirá el texto?

- a) Infinitas.
- b) 3 veces.
- c) 0 veces.

5. `x = 5`
`y = 0`
`z = 0`
`if x > 2:`
 `y = 5`
`else:`
 `z = 5`
`print(z)`

¿Qué valor imprime la variable z?

- a) 0
- b) x e y
- c) 5

FORMA C - Cuestionario de Evaluación Diagnóstica

1. `val = 50`
`cop = val`
`val = 100`
`print(cop)`

¿Qué valor imprime la consola?

- a) Error
- b) 100
- c) 50

2. `nivel = 1`
 `nivel = nivel + 1`
 `print(nivel)`

¿Qué resultado produce este código?

- a) Error matemático.
- b) 1
- c) 2

3. `num = 0`

```
if num < 2:  
    print(num)  
    num += 1
```

¿Qué se imprime en pantalla?

- a) 0
- b) Error de incremento.
- c) 0 1

4. `turnos = 0`

```
while turnos < 5:  
    print("Tu turno")
```

¿Cuántas veces se imprimirá el texto?

- a) 0 veces.
- b) Infinitas.
- c) 5 veces.

5. `v = 8`

```
r1 = 0  
r2 = 0  
if v > 5:  
    r1 = 8  
else:  
    r2 = 8  
print(r2)
```

¿Qué valor imprime la variable r2?

- a) v y r1
- b) 0
- c) 8

Anexo B: Instrumento de Diagnóstico - Iteración 2 (Estructuras de Datos)

Este anexo detalla el diseño de la segunda fase de evaluación, orientada a detectar errores conceptuales en el manejo de listas y diccionarios, con especial énfasis en el modelo de memoria por referencia a objetos mutables y el acceso directo por llave.

Cuestionario Diagnóstico - Forma A

1. Considere un diccionario que asocia a cada letra su tipo, ya sea una vocal o una consonante. Tanto las llaves como los valores son strings.

Escriba la función `obtener_tipo(catalogo, letra)` que reciba como parámetro el diccionario con los tipos de letra y un string con una letra. La función debe retornar el tipo de la letra indicada en el segundo parámetro.

Ejemplos:

```
>>> catalogo = {'a': 'vocal', 'b': 'consonante', 'e': 'vocal'}
>>> print(obtener_tipo(catalogo, 'a'))
vocal
```

- 2.

```
lista1 = [1, 2, 3]
lista2 = lista1
lista2[0] = 99
print(lista1[0])
```

¿Qué valor se imprime en consola?

- a) 99
- b) 1
- c) 0

- 3.

```
edades = [18, 19, 20]
print(edades[2])
```

¿Qué se imprime al ejecutarse esta línea?

- a) 2

- b) 20
- c) 19

4.

```
def configurar_juego():  
    puntos = 100
```

```
configurar_juego()  
print(puntos)
```

¿Qué se imprime al final?

- a) 100
- b) 0
- c) Da un error (NameError).

5.

```
def contar_pares(lista_numeros):  
    contador = 0  
    for num in lista_numeros:  
        if num % 2 == 0:  
            contador += 1  
    return contador
```

```
print(contar_pares([-2, 3, 4, 8]))
```

¿Qué valor imprime la consola?

- a) 3
- b) 0
- c) 1

Cuestionario Diagnóstico - Forma B

1. Considere un diccionario que asocia a cada letra su tipo, ya sea una vocal o una consonante. Tanto las llaves como los valores son strings.

Escriba la función `obtener_tipo(catalogo, letra)` que reciba como parámetro el diccionario con los tipos de letra y un string con una letra. La función debe retornar el tipo de la letra indicada en el segundo parámetro.

Ejemplos:

```
>>> catalogo = {'a': 'vocal', 'b': 'consonante', 'e': 'vocal'}
>>> print(obtener_tipo(catalogo, 'a'))
vocal
```

- 2.

```
grupo1 = [5, 10, 15]
grupo2 = grupo1
grupo2[0] = 42
print(grupo1[0])
```

¿Qué valor se imprime en consola?

- a) 42
- b) 5
- c) 0

- 3.

```
notas = [40, 55, 70]
print(notas[2])
```

¿Qué se imprime al ejecutarse esta línea?

- a) 70
- b) 55
- c) 2

- 4.

```
def cargar_perfil():
    nivel = 5

cargar_perfil()
print(nivel)
```

¿Qué se imprime al final?

- a) 5
- b) 0
- c) Da un error (NameError).

5.

```
def contar_positivos(valores):  
    contador = 0  
    for v in valores:  
        if v > 0:  
            contador += 1  
    return contador  
  
print(contar_positivos([8, -2, 4, 6]))
```

¿Qué valor imprime la consola?

- a) 3
- b) 0
- c) 1

Anexo C: Cuestionario del Sondeo Exploratorio Docente

A continuación, se detalla el instrumento de validación cualitativa enviado al equipo docente a cargo de la asignatura de Programación. El cuestionario fue diseñado para evaluar la percepción de utilidad del panel analítico propuesto y su impacto en la toma de decisiones pedagógicas, estructurándose en tres dimensiones clave.

Dimensión 1: Utilidad del Diagnóstico y Detección de Errores

1. ¿Considera que el formato en el que la herramienta clasifica los resultados (separando los errores estructurales profundos de los simples errores de sintaxis/estrategia) facilita la identificación real de los conceptos que los estudiantes no han comprendido?
2. Al observar las alertas críticas reveladas (como el 40,6 % de error en `if` vs `while` o el masivo 82,5 % de error en el uso de `return`), ¿hubo algún dato que le sorprendiera respecto a lo que usted percibía o diagnosticaba habitualmente en el aula?

Dimensión 2: Toma de Decisiones y Reparación de Misconceptions

3. Basándose en los datos precisos que entrega este reporte, ¿qué contenidos o conceptos específicos decidiría reforzar o deconstruir de manera urgente en la próxima clase antes de una evaluación sumativa?
4. Sabiendo que los estudiantes en Nivel 1 Crítico poseen una falla estructural que no se soluciona simplemente mostrando el código correcto, ¿qué estrategias pedagógicas específicas (ej. ruteo paso a paso manual, instrucción por pares, andamiaje comparativo) implementaría usted para arreglar estas deficiencias puntuales?

Dimensión 3: Proyección Institucional

5. En términos de gestión de tiempo y esfuerzo, ¿considera que tener acceso automático a este nivel de analítica (*Learning Analytics*) optimizaría su preparación de clases y focalización de los contenidos?
6. ¿Recomendaría la integración permanente de esta herramienta como apoyo directo al equipo docente de Programación para diagnosticar tempranamente a las futuras cohortes?

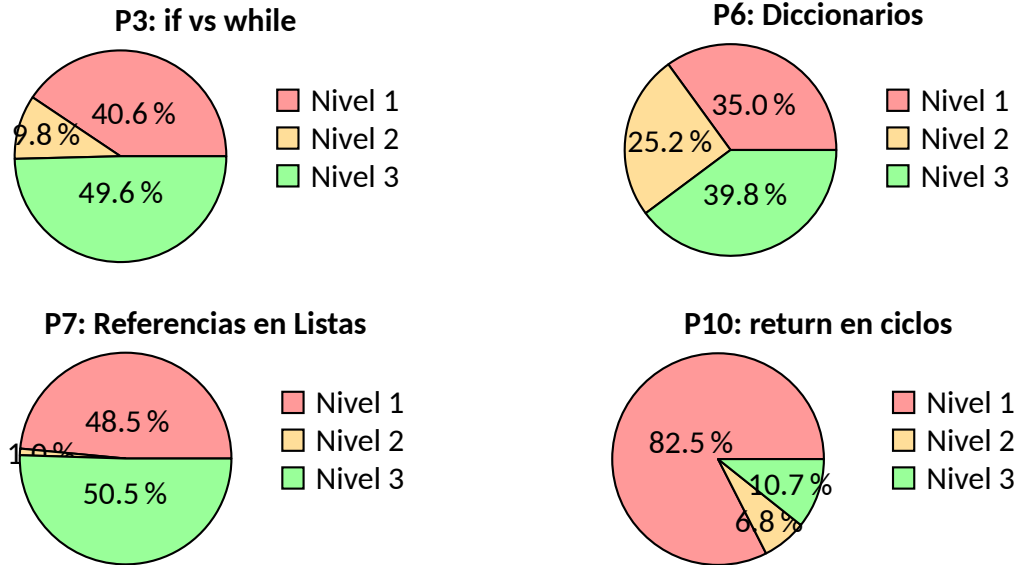
Anexo D: Capturas del Panel Analítico (Dashboard Docente)

Este anexo expone la visualización exacta de los datos que fue entregada a los docentes evaluados durante el Sondeo Exploratorio (Fase 1 y Fase 2). La presentación de este reporte consolidado originó las observaciones de usabilidad mencionadas en el cuerpo del documento, donde los entrevistados sugirieron incluir el código fuente de los reactivos junto a las métricas para facilitar la interpretación clínica inmediata.

1. Matriz de Consolidación Diagnóstica

P	Concepto Evaluado	Nivel 1: Error Crítico	Nivel 2: Error Superficial	Nivel 3: Logro Esperado
P1	Modelo Memoria: Enlaces temporales	21,8 %	1,5 %	76,7 %
P2	Memoria: Actualización Destructiva	2,3 %	0,0 %	97,7 %
P3	Flujo: Condicional vs Iteración (if vs while)	40,6 %	9,8 %	49,6 %
P4	Flujo: "Terminación mágica" de ciclos	16,5 %	0,0 %	83,5 %
P5	Flujo: Exclusión mutua en ramas múltiples	12,0 %	0,0 %	88,0 %
P6	Diccionarios: Búsqueda y acceso por llave	35,0 %	25,2 %	39,8 %
P7	Memoria: Referencias dinámicas en Listas	48,5 %	1,0 %	50,5 %
P8	Listas: Diferenciación Índice vs Contenido	12,6 %	2,9 %	84,5 %
P9	Scope: Inyección global de variables locales	17,5 %	4,9 %	77,7 %
P10	Funciones: Uso del return en un ciclo	82,5 %	6,8 %	10,7 %

2. Gráficas de Distribución de Modelos Mentales y Patrones Críticos



REFERENCIAS BIBLIOGRÁFICAS

- [Alvayay Figueroa, 2024] Alvayay Figueroa, D. F. (2024). Mitigando errores de concepto en estudiantes de cursos introductorios de programación a través de un videojuego. Memoria para optar al título de ingeniero civil en informática, Universidad Técnica Federico Santa María.
- [Ben-Ari, 1998] Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(1):257–261.
- [Bonar y Soloway, 1985] Bonar, J. y Soloway, E. (1985). Preprogramming knowledge: a major source of misconceptions in novice programmers. *Human-Computer Interaction*, 1(2):133–161.
- [Boulay, 1981] Boulay, B. D. (1981). The black box inside the glass box”presenting computing concepts to novices. *Nature*, pp. 237–249.
- [da Silva, 2024] da Silva, E. P. (2024). *Misconceptions in Correct Code: Assisting Instructors and Students by Shedding Light on What is Potentially Overshadowed by Automated Correction*. Tesis doctoral, Universidade Estadual de Campinas (UNICAMP), Instituto de Computação.
- [du Boulay, 1986] du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73.
- [Fincher et al., 2020] Fincher, S., Jeuring, J., Miller, C. S., Donaldson, P., Du Boulay, B., Hauswirth, M., Hellas, A., Hermans, F., Lewis, C., Mühling, A., Pearce, J. L., y Petersen, A. (2020). Notional machines in computing education: The education of attention. En *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 21–50.
- [Fleury, 1991] Fleury, A. E. (1991). Parameter passing: the rules the students construct. En *Proceedings of the twenty-second SIGCSE technical symposium on Computer science education (SIGCSE '91)*, pp. 283–286. Association for Computing Machinery.
- [Gallego-Durán et al., nd] Gallego-Durán, F. J., Compañ-Rosique, P., Villagrà-Arnedo, C. J., García-Sánchez, G. M., y Satorre-Cuerda, R. (n.d.). Modelos mentales erróneos y persistentes en programación.
- [Haberman y Ben-David Kolikant, 2001] Haberman, B. y Ben-David Kolikant, Y. (2001). Activating “black boxes” instead of opening “zipper” - a method of teaching novices basic cs concepts. En *Proceedings of the 6th annual conference on Innovation and technology in computer science education (ITICSE '01)*, pp. 41–44. Association for Computing Machinery.
- [Iturbide, 2021] Iturbide, J. J. V. (2021). Una indagación sobre el comportamiento de lenguajes de programación sencillos basados en bloques.

- [Justi y Van Driel, 2006] Justi, R. y Van Driel, J. (2006). The use of the interconnected model of teacher professional growth for understanding the development of science teachers' knowledge on models and modelling. *Teaching and Teacher Education*, 22(4):437–450.
- [Kaczmarczyk et al., 2010] Kaczmarczyk, L. C., Petrick, E. R., East, J. P., y Herman, G. L. (2010). Identifying student misconceptions of programming. En *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*, pp. 107–111. Association for Computing Machinery.
- [Lister et al., 2004] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., y Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. En *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '04)*, pp. 119–150. Association for Computing Machinery.
- [Ma, 2007] Ma, L. (2007). *Investigating and improving novice programmers' mental models of programming concepts*. Tesis doctoral, University of Strathclyde, UK.
- [Meyer y Land, 2003] Meyer, J. H. F. y Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. Occasional Report 4, Enhancing Teaching-Learning Environments in Undergraduate Courses Project (ETL Project), University of Edinburgh, Edinburgh, UK.
- [Pea, 1986] Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1):25–36.
- [Qian y Lehman, 2017] Qian, Y. y Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24.
- [Sirkiä, 2012] Sirkiä, T. (2012). Recognizing programming misconceptions: An analysis of the data collected from the uuhistle program simulation tool. Tesis de máster, Aalto University, Espoo, Finlandia.
- [Sirkiä y Sorva, 2012] Sirkiä, T. y Sorva, J. (2012). Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. En *Proceedings of the 12th Koli Calling International Conference on Computing Education Research (Koli Calling '12)*, pp. 19–28. Association for Computing Machinery.
- [Sleeman et al., 1986] Sleeman, D., Putnam, R. T., Baxter, J., y Kuspa, L. (1986). Pascal and high school students: A study of errors. *Journal of Educational Computing Research*, 2(1):5–23.
- [Sorva, 2012] Sorva, J. (2012). *Visual program simulation in introductory programming education*. Tesis doctoral, Aalto University, Espoo, Finlandia.

[Vásquez Pinto, 2020] Vásquez Pinto, F. J. (2020). Impacto del uso de un lenguaje basado en bloques en el desarrollo de habilidades de programación. Memoria de titulación para optar al título de ingeniero civil informático, Universidad Técnica Federico Santa María.

[Weintrop y Wilensky, 2015] Weintrop, D. y Wilensky, U. (2015). Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. En *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, pp. 101–110.