Santiago Dynamical System for Convergence Analysis of Software Debloating and Attack-Surface Reduction

Mendoza Delgado Raphael Santiago, Est.[1]; Mendoza Arenas Rubén Dario, Dr.[1]; Delgado Baltazar Marisol Paola, Mg.[1]; Rojas Orbegoso Jorge Luis, Mg.[1]; Artoche Wong Raquel Leonor, Mg.[2]; Yuncar Alvaron Jesús, Dr.[1]; Garrido Huaylinos Luisa Rosario, Dra.[3]; [1]Universidad Nacional del Callao, Perú, rsmendozad@unac.edu.pe, rdmendozaa@unac.edu.pe, mpdelgadob@unac.edu.pe, jlrojaso@unac.edu.pe, jyuncara@unac.edu.pe, [2]Universidad Nacional Federico Villareal, Perú, ratoche@unfv.edu.pe, [3]Universidad Nacional de Educación Enrique Guzmán y Valle, Perú, lgarrido@une.edu.pe.

Abstract—This paper proposes a Santiago Dynamical System (SDS) to analyze the convergence of iterative software debloating and its relationship with attack-surface reduction. A software variant is modeled as a state, a debloating strategy as a transition operator, and code size, dependencies, permitted system calls, reusable-code exposure, and retained functionality as observables. The study follows a quantitative, applied, repeated-measures computational design. Three idealized strategies—static, workload-driven dynamic, and hybrid—were evaluated on a reproducible synthetic benchmark of 90 software systems. The primary outcomes were final structural-complexity index, stabilization iterations, final attack-surface index, and functional-retention percentage. Shapiro–Wilk tests showed that normality was not consistently satisfied; therefore, Friedman tests and Holm-adjusted paired Wilcoxon tests were used. Significant differences were found in structural complexity ($\chi^2 = 138.20$), iterations ($\chi^2 = 119.95$), attack surface ($\chi^2 = 174.07$), and functional retention ($\chi^2 = 112.75$), with $p < 0.001$ and large Kendall effects. The hybrid strategy yielded the lowest median attack-surface index (40.06) and a high median functional retention (97.95%), whereas the dynamic strategy achieved stronger code-size reduction but lower functional retention. The results support a multiobjective interpretation: smaller programs are not automatically safer or more robust. The empirical values are simulated proof-of-concept data and require validation with executable debloating tools, public benchmarks, differential testing, and security-oriented metrics.

Keywords: software debloating, attack surface, discrete dynamical systems, software security, convergence analysis, systems engineering.

Sistema Dinámico de Santiago para Analizar la Convergencia del Desinflado de Software y la Reducción de Superficie de Ataque

Resumen—Se propone un Sistema Dinámico de Santiago (SDS) para analizar la convergencia del desinflado iterativo de software y su relación con la reducción de la superficie de ataque. Cada variante del programa se interpreta como un estado; la estrategia de desinflado, como operador de transición; y el tamaño del código, las dependencias, las llamadas al sistema, la exposición a reutilización de código y la funcionalidad conservada, como observables. Se desarrolló un estudio cuantitativo aplicado, computacional y de medidas repetidas, comparando tres estrategias idealizadas: estática, dinámica guiada por carga e híbrida. Se empleó un banco sintético reproducible de 90 sistemas. Las variables principales fueron complejidad estructural final, iteraciones de estabilización, superficie de ataque final y retención funcional. La normalidad no se cumplió consistentemente; por ello, se aplicaron Friedman y Wilcoxon pareada con corrección de Holm. Se obtuvieron diferencias significativas en complejidad ($\chi^2 = 138{,}20$), iteraciones ($\chi^2 = 119{,}95$), superficie de ataque ($\chi^2 = 174{,}07$) y retención funcional ($\chi^2 = 112{,}75$), con $p < 0{,}001$. La estrategia híbrida presentó la menor superficie de ataque mediana y elevada conservación funcional, mientras que la estrategia dinámica redujo más código, pero sacrificó robustez. Los resultados son una prueba de concepto simulada y requieren validación empírica con herramientas ejecutables y corpus públicos.

Palabras clave: desinflado de software, superficie de ataque, sistemas dinámicos discretos, seguridad de software, análisis de convergencia.

I. Introducción

I-A. Realidad problemática

El crecimiento de las aplicaciones modernas se relaciona con la reutilización de bibliotecas, la incorporación continua de funciones, la compatibilidad con múltiples plataformas y la acumulación de dependencias transitivas. Como resultado, una instalación puede contener código, interfaces y capacidades que nunca serán utilizadas por la carga de trabajo real. Este excedente aumenta el tamaño del artefacto, dificulta el mantenimiento y amplía la cantidad de componentes potencialmente alcanzables por un atacante. El desinflado de software busca retirar funciones, bibliotecas, instrucciones o interfaces innecesarias, preservando un conjunto declarado de requisitos funcionales [1], [2], [3].

Las propuestas existentes operan en distintos niveles. Algunas eliminan código fuente mediante transformaciones y pruebas; otras especializan programas compilados, bibliotecas compartidas, aplicaciones web, firmware, contenedores o llamadas al sistema [4], [5], [6], [8]. Esta diversidad demuestra que el desinflado no es una sola técnica, sino una familia de procesos de especialización condicionados por perfiles de uso, configuraciones, análisis estático, trazas dinámicas y políticas de seguridad.

El problema principal es que una reducción cuantitativa no garantiza por sí misma corrección ni seguridad. Una evaluación comparativa de herramientas reportó dificultades de compatibilidad, madurez, robustez y conservación semántica, y encontró que solo una fracción de los intentos produjo variantes sólidas bajo pruebas diferenciales [10]. De manera semejante, la disminución del número total de gadgets reutilizables puede ocultar gadgets nuevos o cualitativamente más útiles para un atacante; por ello, las métricas de seguridad deben considerar calidad, expresividad y accesibilidad, no únicamente conteos brutos [9].

La literatura suele informar tamaño inicial y final, porcentaje retirado, tiempo o cantidad de vulnerabilidades removidas. Sin embargo, el proceso mismo es iterativo: cada ronda produce una versión, la verificación decide si es admisible y la siguiente ronda parte del resultado aceptado. Falta un marco matemático sencillo que represente la trayectoria completa, caracterice su estabilización y permita comparar operadores con criterios de convergencia, corrección y seguridad. El SDSofrece esta perspectiva mediante órbitas, puntos fijos, tiempos de llegada, atractores y observables discretas [13], [14].

I-B. Marco teórico y antecedentes

El desinflado puede definirse como la especialización de un programa para un conjunto de características o cargas de trabajo. La compilación por piezas mostró que una proporción considerable de bibliotecas y ejecutables permanece sin utilizar en entornos concretos y que parte de ese código puede excluirse en compilación o carga [1]. La especialización mediante propagación de constantes y transformaciones interprocedimentales demostró que las configuraciones conocidas permiten remover rutas inalcanzables [4]. Los enfoques guiados por aprendizaje y búsqueda tratan la selección de transformaciones como un problema de optimización sujeto a pruebas de equivalencia observacional [2], [12].

En artefactos desplegados, la reconstrucción y poda de flujo de control permite producir binarios especializados sin disponer necesariamente del código fuente [3]. En aplicaciones web, el uso observado puede guiar la eliminación de archivos o funciones no ejecutadas y reducir complejidad y exposición histórica a vulnerabilidades [5]. En servidores, la especialización temporal limita las llamadas al sistema según fases de inicialización y servicio, mientras que los generadores automáticos de políticas para contenedores restringen la interfaz con el núcleo siguiendo el principio de mínimo privilegio [6], [7].

Estas técnicas comparten tres tensiones. Primero, una estrategia conservadora retiene código no observado para evitar regresiones, pero deja una superficie mayor. Segundo, una estrategia agresiva alcanza artefactos pequeños, aunque

puede perder comportamientos válidos fuera del perfil de entrenamiento. Tercero, un indicador de tamaño puede mejorar sin que cambie significativamente la seguridad efectiva. Una taxonomía reciente organiza el campo por artefactos de entrada y salida, estrategias y criterios de evaluación, y señala la corrección, la generalización y las métricas como problemas abiertos [11].

I-C. Formulación del Sistema Dinámico de Santiago

Sea $\mathcal{P}$ el conjunto de variantes finitas de software y sea $\mathcal{R}$ un conjunto de requisitos funcionales y de seguridad. Se define

$$\mathcal{P}_{\mathcal{R}} = \{P \in \mathcal{P} : Q_{\mathcal{R}}(P) = 1\}, \tag{1}$$

donde $Q_{\mathcal{R}}$ es un oráculo compuesto por compilación, pruebas funcionales, pruebas diferenciales y restricciones de seguridad. Para una estrategia j, el operador de desinflado es

$$D_j : \mathcal{P}_{\mathcal{R}} \longrightarrow \mathcal{P}_{\mathcal{R}}, \qquad P_{k+1} = D_j(P_k). \tag{2}$$

La sucesión $(P_k)_{k\geq 0}$ es la órbita estructural. El estado no debe reducirse inicialmente a un solo número, porque programas de igual tamaño pueden tener dependencias, interfaces y comportamientos diferentes.

Se adopta la observable vectorial

$$\boldsymbol{z}(P) = \big(c(P), d(P), s(P), g(P), v(P), q(P)\big), \tag{3}$$

donde c es cantidad de instrucciones o unidades de código; d, dependencias; s, llamadas al sistema permitidas; g, exposición ponderada a gadgets; v, componentes vulnerables alcanzables; y $q \in [0, 100]$, retención funcional. El perfil dinámico es

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}(D_j(P_k)). \tag{4}$$

Para comparar trayectorias se define el índice de complejidad relativo

$$C(P_k) = 100\left(,45\frac{c_k}{c_0} + ,20\frac{d_k}{d_0} + ,15\frac{s_k}{s_0} + ,20\frac{g_k}{g_0}\right), \tag{5}$$

y el índice de superficie de ataque

$$A(P_k) = 100\left(,30\frac{s_k}{s_0} + ,25\frac{g_k}{g_0} + ,30\frac{v_k + ,5}{v_0 + ,5} + ,15\frac{d_k}{d_0}\right). \tag{6}$$

Los pesos son explícitos y ajustables; no pretenden constituir un estándar universal. Su finalidad es construir observables reproducibles que eviten identificar seguridad con una sola magnitud [9], [10].

El SDS constituye una formulación aplicada basada en conceptos establecidos de sistemas dinámicos discretos y no pretende introducir una nueva teoría matemática, sino proporcionar un marco común para analizar trayectorias de desinflado de software.

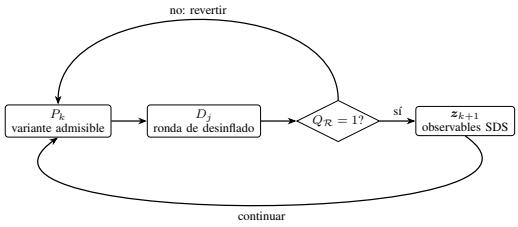


Figura 1. Ciclo de desinflado, verificación y observación en el SDS.

I-D. Definición conceptual de las variables

La variable independiente es la *estrategia iterativa de desinflado*. Se operacionaliza en tres niveles: (i) estático, basado en relaciones de alcance y configuración; (ii) dinámico, basado en cobertura de la carga observada; y (iii) híbrido, que combina información estática, dinámica y políticas de seguridad. Esta clasificación representa familias metodológicas documentadas en la literatura, no implementaciones concretas de una herramienta particular [3], [11].

La variable dependiente es la *convergencia segura del software desinflado*, entendida como la capacidad de alcanzar una configuración estable con baja complejidad y baja superficie de ataque sin perder las funciones requeridas. Sus dimensiones son: complejidad final C_f; tiempo de llegada τ; superficie de ataque final A_f; retención funcional q_f; consultas al oráculo; y tiempo computacional.

I-E. Planteamiento de hipótesis

La hipótesis general sostiene que la estrategia de desinflado produce diferencias significativas en la convergencia segura del SDS. Se formularon las siguientes hipótesis específicas:

- H_1: las estrategias difieren significativamente en el índice de complejidad estructural final.
- H_2: las estrategias difieren significativamente en las iteraciones requeridas para alcanzar un punto fijo.
- H_3: las estrategias difieren significativamente en el índice final de superficie de ataque.
- H_4: las estrategias difieren significativamente en el porcentaje de funcionalidad retenida.

Las hipótesis nulas establecen igualdad de distribuciones entre las tres condiciones. Se adoptó $\alpha = 0{,}05$ y se controló el error familiar en comparaciones posteriores.

II. Materiales y Métodos

II-A. Enfoque, tipo, alcance y diseño

El estudio tuvo enfoque cuantitativo, tipo aplicado y alcance explicativo-comparativo. Se utilizó un diseño computacional de medidas repetidas: cada sistema inicial fue sometido a las tres estrategias, de modo que el propio programa actuó como bloque. El diseño permite estudiar diferencias intra-sistema y reduce la variabilidad atribuible al tamaño inicial. La sección empírica constituye una prueba de concepto simulada; no representa la ejecución de herramientas reales ni resultados industriales.

Cuadro I
OPERACIONALIZACIÓN RESUMIDA DE LAS VARIABLES.

Variable	Dimensión	Indicador
Estrategia	Tipo de análisis	Estática, dinámica, híbrida
Convergencia	Complejidad	C_f de (5)
Convergencia	Estabilización	$\tau = \min\{k : P_{k+1} = P_k\}$
Seguridad	Superficie	A_f de (6)
Corrección	Retención	Pruebas superadas q_f (%)
Costo	Consultas	Ejecuciones de $Q_{\mathcal{R}}$
Costo	Tiempo	Segundos simulados

II-B. Población, muestra y unidad de análisis

La población conceptual estuvo formada por aplicaciones finitas que pueden especializarse para un conjunto de requisitos verificables. Se generó una muestra sintética de 90 sistemas con tamaños iniciales entre 24,689 y 377,165 unidades de código, entre 10 y 240 dependencias, entre 40 y 320 llamadas al sistema y entre 1 y 30 componentes vulnerables potencialmente alcanzables. La unidad de análisis fue la trayectoria estrategia–sistema, obteniéndose 270 órbitas.

La muestra se eligió para proporcionar variabilidad suficiente en tamaño y dificultad y permitir contrastes pareados. La semilla pseudoaleatoria fue 20260802. Los artefactos sintéticos no contienen datos personales, código propietario ni vulnerabilidades explotables.

II-C. Tratamientos experimentales

La condición estática mantuvo mayor cantidad de código ante incertidumbre y priorizó retención funcional. La condición dinámica eliminó agresivamente elementos no observados en la carga, con riesgo de sobreajuste. La condición híbrida combinó alcance estático, evidencia dinámica y restricciones sobre dependencias, llamadas y componentes vulnerables. Estos perfiles se inspiraron en diferencias generales entre especialización estática, guiada por uso y combinada [1], [3], [6].

II-D. Modelo de simulación

Para cada sistema i, estrategia j, observable decreciente r e iteración k, se utilizó una recurrencia afín con piso:

$$z_{i,j,k+1}^{(r)} = \max\Big\{ m_{ij}^{(r)},\ \Big\lceil m_{ij}^{(r)} + c_j^{(r)}\big(z_{i,j,k}^{(r)} - m_{ij}^{(r)}\big) + \varepsilon_{i,j,k} \Big\rceil \Big\}. \tag{7}$$

con $0 < c_j^{(r)} < 1$, piso $m_{ij}^{(r)}$ y perturbación pequeña $\varepsilon_{i,j,k}$. El piso representa el núcleo funcional irreductible bajo la estrategia y la carga. Se impuso monotonía no creciente en código, dependencias, llamadas, gadgets y componentes vulnerables. La funcionalidad retenida se generó como una variable acotada cuya mediana fue mayor en la condición estática, intermedia en la híbrida y menor en la dinámica.

Las medianas objetivo de razón final de código fueron aproximadamente 0.62, 0.40 y 0.48 para las condiciones estática, dinámica e híbrida. La estrategia híbrida recibió menores razones finales en dependencias, llamadas, gadgets y vulnerabilidades para representar una optimización orientada a seguridad. Las consultas crecieron con τ, el logaritmo del tamaño inicial y un factor de dificultad compartido. Estas diferencias fueron definidas mediante perfiles parametrizados de simulación; por tanto, los contrastes posteriores evalúan la capacidad del SDS para distinguir dichos perfiles y no constituyen evidencia empírica de superioridad entre herramientas reales de desinflado. Esta construcción produce datos pareados, asimetría y costos heterogéneos, rasgos frecuentes al evaluar algoritmos de Ingeniería de Software [15].

Algorithm 1 Generación de una trayectoria SDS de desinflado.

Entrada: Estado P_0, estrategia j, pisos $\boldsymbol{m}_{ij}$, máximo K
Salida: Órbita observable $(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_\tau)$

$k \leftarrow 0$; verificar $Q_{\mathcal{R}}(P_0) = 1$
while $k < K$ y P_k no sea punto fijo **do**
  Proponer $\widetilde{P} \leftarrow D_j(P_k)$
  Calcular $Q_{\mathcal{R}}(\widetilde{P})$
  if $Q_{\mathcal{R}}(\widetilde{P}) = 1$ **then**
    $P_{k+1} \leftarrow \widetilde{P}$
  else
    Revertir o conservar $P_{k+1} \leftarrow P_k$
  end if
  Registrar $\boldsymbol{z}_{k+1}$, C_{k+1} y A_{k+1}
  $k \leftarrow k + 1$
end while
$\tau \leftarrow k$
return $(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_\tau)$

II-E. Instrumentos, procedimiento y reproducibilidad

Se diseñó una matriz de registro con identificador, estrategia, observables iniciales y finales, complejidad, superficie, funcionalidad, iteraciones, consultas y tiempo. El procedimiento comprendió: (1) generación de sistemas; (2) asignación de dificultad compartida; (3) ejecución de las tres trayectorias; (4) verificación de cotas y monotonía; (5) cálculo de índices; (6) análisis estadístico; y (7) elaboración de visualizaciones.

La implementación de referencia se planteó en Python con arreglos numéricos y rutinas científicas ampliamente documentadas [16], [17]. Para una replicación real deben publicarse versiones de herramientas, compiladores, cargas, scripts de pruebas, tiempos límite y resultados fallidos. Los errores de compilación y las variantes no robustas deben conservarse como resultados, no descartarse silenciosamente [10].

II-F. Análisis estadístico

Se calcularon mediana y rango intercuartílico. La normalidad se examinó con Shapiro–Wilk [18]. Debido al diseño pareado, la discreción de las iteraciones y el incumplimiento parcial de normalidad, las diferencias globales se contrastaron con Friedman [19]; el tamaño del efecto se expresó mediante W de Kendall. Las comparaciones por pares utilizaron Wilcoxon de rangos con signo y corrección secuencial de Holm [20], [21]. Los efectos globales se consideraron grandes desde $W \geq 0{,}50$.

II-G. Validez y delimitaciones

La semilla fija, el diseño pareado y las fórmulas explícitas fortalecen reproducibilidad y validez interna. La validez de

constructo está limitada por los pesos de C y A y por el uso de observables agregadas. La validez externa es restringida porque el banco es sintético. Los resultados sirven para comprobar el marco SDS y diseñar un protocolo inferencial, no para afirmar superioridad definitiva de herramientas reales. En futuras validaciones se recomienda realizar análisis de sensibilidad de los pesos utilizados en los índices C y A para determinar la robustez de las conclusiones frente a distintas parametrizaciones.

III. Resultados

III-A. Resultados teóricos de convergencia

Proposición 1 (terminación por descenso bien fundado). Sea $D : \mathcal{P}_{\mathcal{R}} \to \mathcal{P}_{\mathcal{R}}$ y sea $L : \mathcal{P}_{\mathcal{R}} \to \mathbb{N}$ una medida tal que

$$D(P) \neq P \Longrightarrow L(D(P)) < L(P). \tag{8}$$

Entonces toda órbita alcanza un punto fijo en un número finito de iteraciones.

Demostración. Mientras P_k no sea fijo, $L(P_{k+1}) < L(P_k)$. No existe una sucesión infinita estrictamente decreciente en $\mathbb{N}$; por el principio del buen orden, existe τ tal que $D(P_\tau) = P_\tau$. □

El resultado exige que cada cambio aceptado disminuya una medida bien fundada. Un punto fijo es irreducible respecto del repertorio de transformaciones y del oráculo utilizados, pero no necesariamente mínimo global. Además, puede ser específico de la carga que definió $Q_{\mathcal{R}}$ [2], [3].

Proposición 2 (cota afín). Si una observable escalar satisface $z_{k+1} \leq cz_k + b$, con $0 < c < 1$, entonces

$$z_k \leq c^k z_0 + \frac{b(1-c^k)}{1-c}. \tag{9}$$

La sustitución recursiva y la suma geométrica demuestran (9). El parámetro c mide contracción por ronda y $b/(1-c)$ aproxima el nivel residual irreductible.

Proposición 3 (no equivalencia entre tamaño y seguridad). No existe, en general, una función estrictamente creciente f tal que $A(P) = f(C(P))$ para todas las variantes admisibles. En efecto, dos variantes pueden tener el mismo tamaño relativo y distinta accesibilidad de llamadas peligrosas, vulnerabilidades o gadgets cualitativos. Esta observación coincide con evaluaciones que cuestionan la inferencia "menos código implica necesariamente más seguridad" [9], [10].

III-B. Resultados descriptivos

El Cuadro II muestra los resultados. La condición estática conservó mayor complejidad, pero alcanzó la mayor retención funcional. La dinámica produjo la mayor reducción mediana de código (59.88 %), aunque su retención funcional descendió a 91.25 %. La híbrida obtuvo el menor índice mediano de superficie de ataque (40.06), una complejidad final de 43.81 y retención de 97.95 %. El patrón evidencia un compromiso entre agresividad, seguridad y corrección.

La Figura 2 representa trayectorias medianas idealizadas de complejidad. La condición estática llega pronto a un piso alto; la dinámica continúa reduciendo durante más iteraciones; y la híbrida alcanza un piso ligeramente menor que la dinámica con un tiempo intermedio. La estabilización no debe interpretarse como optimalidad global, sino como ausencia de nuevas transformaciones aceptadas bajo el operador.

Cuadro II
Resultados descriptivos: mediana [RIQ], $n = 90$.

Indicador	Estática	Dinámica	Híbrida
C_f	66.46 [63.47–68.67]	46.57 [43.21–49.40]	43.81 [41.26–46.69]
Iteraciones	10 [8–11]	16 [13–18]	12 [11–13]
A_f	72.02 [69.87–76.60]	53.03 [49.68–57.57]	40.06 [36.66–44.93]
Retención (%)	99.25 [98.71–99.67]	91.25 [88.42–95.65]	97.95 [97.18–98.76]
Consultas	202.5 [179.2–232.8]	411 [344.2–469.5]	390 [335.8–436.8]
Tiempo (s)	47.63 [28.38–67.69]	120.18 [82.14–192.99]	145.51 [103.34–201.67]
Red. código (%)	37.56 [32.25–42.68]	59.88 [54.44–65.71]	51.46 [46.35–56.28]
Red. ataque (%)	27.98 [23.40–30.13]	46.97 [42.43–50.32]	59.94 [55.07–63.34]

Cuadro III
Prueba de Shapiro–Wilk por estrategia.

Variable	Estrategia	W	p
C_f	Estática	0.979	0.142
	Dinámica	0.990	0.699
	Híbrida	0.989	0.690
Iteraciones	Estática	0.971	0.042
	Dinámica	0.982	0.257
	Híbrida	0.964	0.014
A_f	Estática	0.985	0.411
	Dinámica	0.973	0.058
	Híbrida	0.988	0.569
Retención	Estática	0.944	$< 0{,}001$
	Dinámica	0.981	0.213
	Híbrida	0.978	0.127

III-C. Análisis de normalidad

El Cuadro III presenta Shapiro–Wilk. La normalidad no se rechazó para C_f ni para A_f en las tres estrategias. No obstante, las iteraciones fueron no normales en las condiciones estática e híbrida, y la retención funcional fue claramente no normal en la condición estática debido al límite superior de 100 %. La mezcla de variables discretas y acotadas, junto con el incumplimiento en varias celdas, justificó mantener el plan no paramétrico. La decisión también evita asumir homogeneidad de varianzas y normalidad multivariada en un diseño de medidas repetidas [15].

III-D. Resultados inferenciales y contrastación

Friedman rechazó las cuatro hipótesis nulas (Cuadro IV). Todos los efectos fueron grandes. El mayor correspondió a superficie de ataque ($W = 0{,}967$), lo que indica un ordenamiento muy consistente dentro de cada sistema. La complejidad presentó $W = 0{,}768$; las iteraciones, $W = 0{,}666$; y la retención funcional, $W = 0{,}626$. Debe enfatizarse que estas diferencias estadísticas corresponden a los perfiles parametrizados del experimento simulado y no deben interpretarse como evidencia de superioridad entre implementaciones reales de las estrategias evaluadas.

Las comparaciones posteriores (Cuadro V) mostraron que todas las diferencias permanecieron significativas después de Holm. En complejidad, la dinámica y la híbrida fueron menores

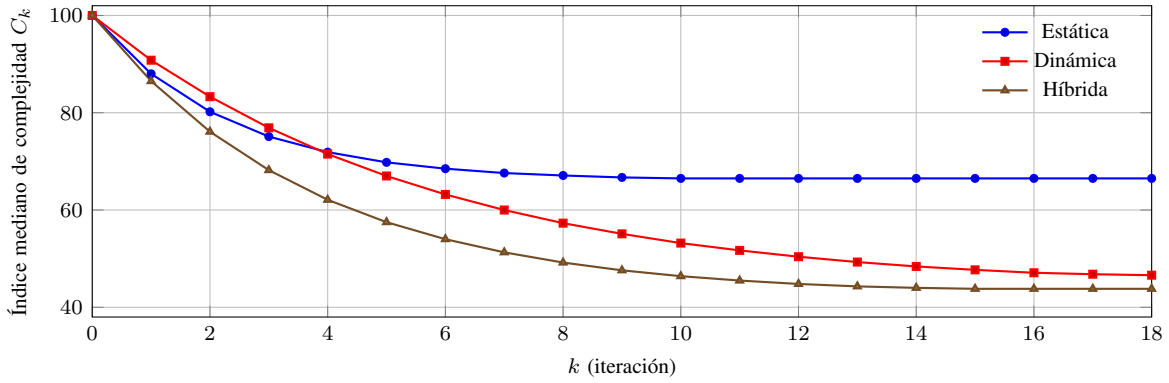


Figura 2. Trayectorias medianas simuladas del índice de complejidad.

Cuadro IV
CONTRASTES GLOBALES DE FRIEDMAN.

Variable	$\chi^2(2)$	p	Kendall W
Complejidad C_f	138.20	$< 0{,}001$	0.768
Iteraciones τ	119.95	$< 0{,}001$	0.666
Superficie A_f	174.07	$< 0{,}001$	0.967
Retención q_f	112.75	$< 0{,}001$	0.626

Cuadro V
WILCOXON PAREADA CON CORRECCIÓN DE HOLM.

Variable	Comparación	V	p_{Holm}
C_f	Est.–Din.	0.0	$< 0{,}001$
	Est.–Híb.	0.0	$< 0{,}001$
	Din.–Híb.	1170.0	$< 0{,}001$
τ	Est.–Din.	2.0	$< 0{,}001$
	Est.–Híb.	400.0	$< 0{,}001$
	Din.–Híb.	192.5	$< 0{,}001$
A_f	Est.–Din.	2.0	$< 0{,}001$
	Est.–Híb.	0.0	$< 0{,}001$
	Din.–Híb.	10.0	$< 0{,}001$
q_f	Est.–Din.	23.0	$< 0{,}001$
	Est.–Híb.	269.0	$< 0{,}001$
	Din.–Híb.	178.0	$< 0{,}001$

que la estática; además, la híbrida fue modestamente menor que la dinámica ($p_{Holm} < 0{,}001$). En iteraciones, la dinámica requirió más rondas, la híbrida ocupó una posición intermedia y la estática estabilizó antes debido a su piso conservador.

En superficie de ataque se obtuvo un orden inequívoco: híbrida $<$ dinámica $<$ estática. En retención funcional ocurrió el patrón estática $>$ híbrida $>$ dinámica. Por tanto, se aceptó H_1, pues la complejidad final difirió; se aceptó H_2, por diferencias en tiempo de llegada; se aceptó H_3, por diferencias en superficie; y se aceptó H_4, por diferencias en conservación funcional. En conjunto se respaldó la hipótesis general.

III-E. Análisis multiobjetivo

Los resultados no identifican una estrategia universalmente dominante en todos los criterios. La estática minimizó iteraciones y maximizó retención, pero dejó mayor complejidad y superficie. La dinámica maximizó reducción de código, aunque disminuyó funcionalidad y elevó consultas. La híbrida minimizó

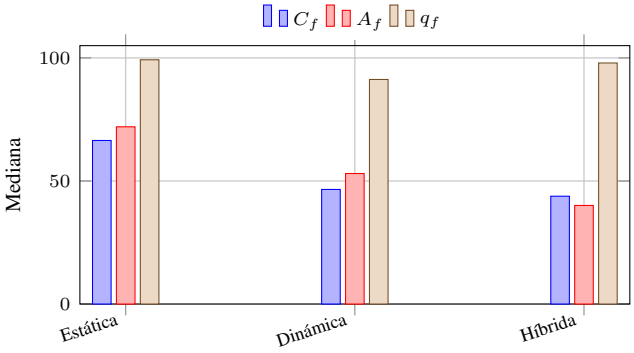


Figura 3. Medianas de complejidad, superficie y retención funcional.

superficie con alta retención, pero presentó el mayor tiempo mediano por el costo de combinar análisis y validaciones.

Para representar este compromiso se propone el conjunto factible

$$\mathcal{F} = \{P \in \mathcal{P}_{\mathcal{R}} : C(P) \le C_{\text{máx}}, \quad A(P) \le A_{\text{máx}},\ q(P) \ge q_{\text{mín}}\}. \tag{10}$$

y una selección por dominancia de Pareto. Una variante domina a otra si no es peor en complejidad, superficie, funcionalidad y costo, y es estrictamente mejor en al menos una dimensión. Este criterio evita convertir observables heterogéneas en una sola puntuación arbitraria.

IV. APLICACIÓN EN INGENIERÍA DE SISTEMAS

IV-A. Arquitectura de una herramienta basada en SDS

La aplicación propuesta es un módulo de evaluación y gobierno del desinflado dentro del ciclo de entrega de software. El módulo recibe un artefacto, un perfil de uso, una política de seguridad y una batería de pruebas; ejecuta rondas de especialización; registra la órbita; y produce un reporte de convergencia. No sustituye al reductor, sino que normaliza observables y permite comparar herramientas heterogéneas.

El registro mínimo por iteración incluye hash del artefacto, configuración, versión del reductor, pruebas superadas, cobertura, código removido, dependencias, llamadas permitidas, vulnerabilidades alcanzables, métricas cualitativas de gadgets,

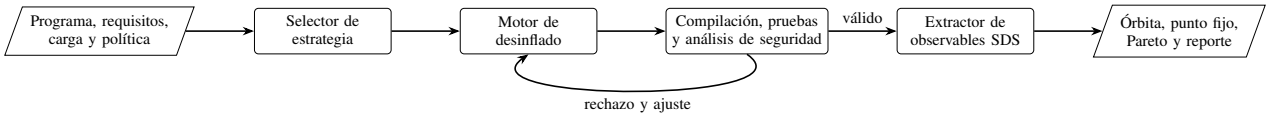


Figura 4. Arquitectura conceptual para evaluar desinflado mediante SDS.

tiempo y memoria. La trazabilidad es indispensable porque dos artefactos con iguales índices pueden diferir semánticamente y porque una reducción puede ser válida para una carga, pero no para otra [3], [10].

IV-B. Escenarios de uso

En DevSecOps, el módulo puede comparar una versión general con variantes especializadas para servicios concretos. En contenedores, permite registrar cómo se restringen binarios, bibliotecas y llamadas al sistema antes de desplegar una imagen. En aplicaciones web, puede estudiar archivos o funciones no requeridos por determinados perfiles. En sistemas embebidos y firmware, sirve para seguir rondas de poda bajo pruebas de arranque, controladores y servicios esenciales [5], [7], [8].

En mantenimiento de legado, el SDS ayuda a distinguir tres situaciones: convergencia rápida a un piso alto, indicativa de un operador conservador; convergencia lenta con grandes reducciones, propia de una búsqueda agresiva; y oscilaciones o retrocesos, señal de transformaciones incompatibles, oráculos inestables o reparaciones que reintroducen código. En este último caso el estado debe ampliarse y la política de aceptación debe impedir ciclos improductivos.

IV-C. Criterios de decisión

La adopción organizacional no debe basarse solo en el porcentaje de código retirado. Se recomienda establecer umbrales de corrección, robustez, desempeño y seguridad antes de optimizar tamaño. Una regla de aceptación puede ser

$$Q_{\mathcal{R}}(P) = 1 \iff q(P) \geq 99{,}5, \quad \Delta T(P) \leq 5\,\%, \quad A(P) \leq A_{\text{máx}}. \tag{11}$$

complementada con pruebas diferenciales y fuzzing. Los umbrales dependen de criticidad y no deben transferirse sin análisis de riesgo.

La reducción de gadgets requiere métricas cualitativas: disponibilidad de operaciones, expresividad, rutas alcanzables y posible aparición de nuevos gadgets. El conteo simple puede utilizarse como indicador exploratorio, pero no como prueba suficiente de seguridad [9]. De igual modo, eliminar componentes asociados con vulnerabilidades históricas es útil, aunque debe verificarse que las rutas vulnerables hayan quedado realmente inaccesibles [5].

IV-D. Protocolo para validación empírica

Una validación posterior debe seleccionar programas abiertos de complejidad baja, media y alta; ejecutar al menos una herramienta por familia; y conservar todos los intentos, incluidos errores y tiempos agotados. Cada versión debe someterse a pruebas originales, pruebas generadas, entradas fuera del perfil y análisis de seguridad. La evaluación comparativa reciente demuestra que las tasas de éxito, robustez y mejora de seguridad pueden ser mucho menores que las sugeridas por estudios aislados, por lo que la replicación debe priorizar exactitud sobre tasas de reducción [10].

Cuadro VI
LISTA DE COMPROBACIÓN PARA UNA REPLICACIÓN REAL.

Elemento	Información requerida
Corpus	Repositorio, licencia, lenguaje, tamaño y complejidad.
Requisitos	Funciones obligatorias y perfiles de uso.
Herramientas	Versión, configuración, transformaciones y fallos.
Entorno	Hardware, SO, compilador y dependencias.
Corrección	Pruebas, cobertura, fuzzing y comparación diferencial.
Seguridad	CVE, llamadas, dependencias y métricas de gadgets.
Costo	Tiempo, memoria, consultas e iteraciones.
Reproducción	Semillas, scripts, datos por caso y artefactos.

V. DISCUSIÓN

La formulación dinámica aporta una lectura distinta a la evaluación tradicional. En vez de observar únicamente la diferencia entre P_0 y P_τ, registra la órbita, la velocidad de contracción, el costo de cada ronda y el momento en que el operador deja de producir mejoras. Esto permite detectar reductores que alcanzan resultados similares mediante trayectorias muy diferentes y fundamentar reglas de detención por ganancia marginal.

La contribución principal del SDS consiste, por tanto, en integrar dentro de un mismo marco la trayectoria del proceso, el punto fijo alcanzado, la velocidad de convergencia, la superficie de ataque, la retención funcional y el costo computacional, permitiendo una evaluación multiobjetivo del desinflado.

El resultado híbrido concuerda con la idea de que combinar análisis puede orientar la reducción hacia componentes con mayor relevancia de seguridad. La especialización temporal de llamadas y la poda de código o dependencias afectan dimensiones distintas de la superficie [6], [7]. No obstante, el mayor tiempo mediano muestra que una superficie menor puede exigir verificación adicional. En contextos de integración continua, este costo puede reservarse para compilaciones nocturnas o versiones de producción.

La condición dinámica mostró la mayor reducción de código y la menor retención funcional. Este resultado representa el riesgo de que un perfil de entrenamiento incompleto clasifique como innecesario un comportamiento válido. Los marcos que infieren código complementario o combinan análisis estático intentan reducir este problema [3]. La evaluación debe incluir entradas fuera del perfil y no limitarse a repetir los mismos casos empleados para guiar la eliminación.

La condición estática estabilizó con menos iteraciones, pero no fue la mejor en complejidad ni seguridad. La convergencia rápida no equivale a mayor eficacia: puede reflejar un espacio de transformaciones reducido o reglas muy conservadoras. Por ello, τ debe interpretarse junto con el atractor alcanzado y el costo por iteración.

Los resultados apoyan la advertencia de que tamaño, seguridad y corrección son constructos relacionados pero no equivalentes. La evidencia empírica disponible muestra que el desinflado puede retirar vulnerabilidades y reducir interfaces en escenarios específicos [5], [6]; sin embargo, las comparaciones amplias encuentran beneficios menos uniformes y problemas de solidez [10]. El SDS no resuelve esta tensión, pero la hace explícita mediante un estado vectorial y una frontera de Pareto.

Entre las amenazas a la validez se encuentran la simulación, los pesos de los índices, la ausencia de errores reales de herramientas y la representación simplificada de la seguridad. Las pruebas de significación reflejan diferencias construidas por los parámetros del modelo; por tanto, no deben interpretarse como evidencia causal sobre implementaciones existentes. La contribución de esta prueba de concepto es mostrar un diseño computacional reproducible que puede posteriormente trasladarse y validarse con datos, programas y herramientas reales.

La formulación también abre preguntas matemáticas. Puede estudiarse la existencia de atractores dependientes de carga, la estabilidad frente a cambios del oráculo, la equivalencia entre puntos fijos producidos por operadores distintos y la posibilidad de ciclos cuando se alternan eliminación y reparación. En estados parcialmente ordenados, la convergencia podría analizarse mediante medidas bien fundadas vectoriales o funciones de Lyapunov discretas.

VI. Conclusiones

Se diseñó un SDS para representar el desinflado como una dinámica sobre variantes admisibles de software. El operador corresponde a una estrategia de especialización, el oráculo preserva requisitos y el estado observable reúne código, dependencias, llamadas, exposición a gadgets, vulnerabilidades y retención funcional. Esta representación evita reducir el fenómeno exclusivamente al tamaño.

Se demostró que un operador que disminuye estrictamente una medida natural bien fundada alcanza un punto fijo en tiempo finito. También se obtuvo una cota afín para observables contractivas y se justificó que no existe una equivalencia general entre complejidad y superficie de ataque.

En la prueba de concepto con 90 sistemas sintéticos, la normalidad no se cumplió consistentemente. Bajo los perfiles parametrizados de simulación, Friedman encontró diferencias significativas y efectos grandes en complejidad, iteraciones, superficie y retención. Las comparaciones de Wilcoxon-Holm respaldaron las cuatro hipótesis específicas y la hipótesis general dentro del escenario simulado.

En el escenario simulado, la estrategia híbrida presentó la menor superficie de ataque mediana y alta retención funcional; la dinámica retiró más código, pero conservó menos funcionalidad; y la estática convergió más rápido a un piso mayor. En consecuencia, la selección debe plantearse como optimización multiobjetivo y no como maximización aislada del porcentaje retirado.

El estudio no afirma que una herramienta real sea superior. Los datos son simulados y deberán validarse en trabajos posteriores, corpus abiertos, herramientas ejecutables, pruebas diferenciales, fuzzing y métricas cualitativas de seguridad. Se recomienda publicar órbitas completas, fallos, configuraciones y artefactos para que las comparaciones sean auditables.

Como línea futura se propone un SDS estocástico para oráculos inestables, un SDS vectorial con orden de Pareto, selectores adaptativos de estrategia y análisis de cuencas dependientes de la carga de trabajo. Estas extensiones pueden integrar matemática discreta, seguridad de software y gestión de sistemas dentro de un marco común.

VI-A. Recomendaciones de implementación y trabajo futuro

Para una implementación institucional se recomienda comenzar con servicios no críticos, artefactos reproducibles y baterías de pruebas maduras. El primer objetivo no debe ser retirar la mayor cantidad posible de código, sino comprobar que el oráculo distingue correctamente entre una variante admisible y otra que ha perdido comportamiento. La cobertura debe complementarse con casos negativos, límites, errores esperados y entradas generadas, porque una carga habitual rara vez representa por sí sola el dominio operacional completo.

Conviene separar tres registros: la trayectoria del reductor, la trayectoria de validación y la trayectoria de seguridad. La primera contiene las transformaciones y tamaños; la segunda, pruebas superadas, fallos y tiempos; y la tercera, dependencias, llamadas, vulnerabilidades y métricas cualitativas. Esta separación permite determinar si una mejora aparente procede de retirar código, de cambiar la política de ejecución o simplemente de modificar la forma de medir.

El despliegue debe conservar siempre el artefacto original, la configuración de reducción y una ruta de reversión. Cada variante especializada debe identificarse mediante hash y vincularse con la carga y la política que justificaron su creación. Cuando cambien requisitos, bibliotecas o patrones de uso, la órbita debe recalcularse; un punto fijo anterior puede dejar de ser admisible bajo un nuevo oráculo.

En el análisis estadístico futuro se recomienda aumentar el número de programas y repetir cada ejecución para separar variabilidad algorítmica de variabilidad del entorno. Los tiempos agotados y los errores de herramienta deben modelarse explícitamente. Además de Friedman y Wilcoxon, pueden emplearse modelos mixtos o análisis de supervivencia cuando existan censuras por tiempo límite, siempre que sus supuestos sean verificados.

Desde el punto de vista matemático, debe estudiarse si dos operadores diferentes poseen atractores equivalentes bajo una relación de equivalencia funcional. También interesa caracterizar condiciones bajo las cuales la alternancia de operadores conserva el descenso de una medida bien fundada. Si una fase de reparación reintroduce componentes, pueden

aparecer ciclos; en ese caso será necesario utilizar una función de Lyapunov vectorial o una política que penalice estados previamente visitados.

Finalmente, la superficie de ataque debe tratarse como un constructo multidimensional y dependiente del adversario. Las futuras réplicas deberían reportar no solo cantidades eliminadas, sino también accesibilidad, privilegios, rutas de explotación, calidad de gadgets y vulnerabilidades realmente neutralizadas. Esta recomendación es coherente con la evidencia que advierte que las métricas simples pueden sobreestimar los beneficios de seguridad del desinflado [9], [10].

REFERENCIAS

[1] A. Quach, A. Prakash, and L. K. Yan, "Debloating software through piece-wise compilation and loading," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 869–886. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/quach

[2] K. Heo, W. Lee, P. Pashakhanloo, and M. Naik, "Effective program debloating via reinforcement learning," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, 2018, pp. 380–394, doi: 10.1145/3243734.3243838.

[3] C. Qian, H. Hu, M. Alharthi, P. H. Chung, T. Kim, and W. Lee, "RAZOR: A framework for post-deployment software debloating," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 1733–1750. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/qian

[4] H. Sharif, M. Abubakar, A. Gehani, and F. Zaffar, "TRIMMER: Application specialization for code debloating," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Software Engineering*, 2018, pp. 329–339, doi: 10.1145/3238147.3238160.

[5] B. A. Azad, P. Laperdrix, and N. Nikiforakis, "Less is more: Quantifying the security benefits of debloating web applications," in *Proc. 28th USENIX Security Symp.*, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/azad

[6] S. Ghavamnia, T. Palit, S. Mishra, and M. Polychronakis, "Temporal system call specialization for attack surface reduction," in *Proc. 29th USENIX Security Symp.*, 2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/ghavamnia

[7] N. DeMarinis, K. Sun, R. Fonseca, and R. Sekar, "sysfilter: Automated system call filtering for commodity software," in *Proc. 23rd Int. Symp. Research in Attacks, Intrusions and Defenses*, 2020. [Online]. Available: https://www.usenix.org/system/files/raid20-demarinis.pdf

[8] J. Christensen, I. M. Anghel, R. Taglang, M. Chiroiu, and R. Sion, "DECAF: Automatic, adaptive de-bloating and hardening of COTS firmware," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 1713–1730. [Online]. Available: https://www.usenix.org/system/files/sec20-christensen.pdf

[9] M. D. Brown and S. Pande, "Is less really more? Towards better metrics for measuring security improvements realized through software debloating," in *Proc. 12th USENIX Workshop Cyber Security Experimentation and Test*, 2019. [Online]. Available: https://www.usenix.org/conference/cset19/presentation/brown

[10] M. D. Brown, A. Meily, B. Fairservice, A. Sood, J. Dorn, E. Kilmer, and R. Eytchison, "A broad comparative evaluation of software debloating tools," in *Proc. 33rd USENIX Security Symp.*, 2024, pp. 3927–3943. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/brown

[11] M. Alhanahnah, Y. Boshmaf, and A. Gehani, "SoK: Software debloating landscape and future directions," in *Proc. 3rd Int. Workshop on Foundations of Software Engineering*, 2024, doi: 10.1145/3689937.3695792.

[12] N. Harrand, S. Allier, M. Rodriguez-Cancio, M. Monperrus, and B. Baudry, "Program debloating via stochastic optimization," in *Proc. ACM/IEEE 42nd Int. Conf. Software Engineering*, 2020, doi: 10.1145/3377816.3381739.

[13] K. T. Alligood, T. D. Sauer, and J. A. Yorke, *Chaos: An Introduction to Dynamical Systems*. New York, NY, USA: Springer, 1996.

[14] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, 2nd ed. Boulder, CO, USA: Westview Press, 2003.

[15] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014, doi: 10.1002/stvr.1486.

[16] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[17] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[18] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, nos. 3–4, pp. 591–611, 1965.

[19] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Statistical Assoc.*, vol. 32, no. 200, pp. 675–701, 1937.

[20] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[21] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian J. Statistics*, vol. 6, no. 2, pp. 65–70, 1979.