https://doi.org/10.69639/arandu.v13i2.2258

Desarrollo Aumentado por Agencia (DAA): Un Marco Metodológico para la Integración de Agentes de Inteligencia Artificial en el Ciclo de Vida de Desarrollo del Software

Agency Augmented Development (AAD): A Methodological Framework for Integrating Artificial Intelligence Agents into the Software Development Life Cycle

Jaime Vladimir Sancho Zurita
jsancho@itsjapon.edu.ec
https://orcid.org/0000-0002-5915-2100
Instituto Tectológico Universitario Japón
Ecuador- Quito

Diego Efrain Quinga Jerez
diego040983@gmail.com
https://orcid.org/0009-0000-2822-6042
Investigador Independiente
Ecuador- Quito

Edison Rolando Moyolema Moyolema
edisonmoyolema@hotmail.com
https://orcid.org/0009-0007-2105-3071
Investigador Independiente
Ecuador- Quito

Jairo Mauricio Benavides Castillo
Jairo.benavides@byz.com.ec
https://orcid.org/0009-0003-8087-0902
Investigador Independiente
Ecuador- Quito

Artículo recibido: 10 abril 2026- Aceptado para publicación:16 mayo 2026
Conflictos de intereses: Ninguno que declarar.

RESUMEN

La irrupción de la inteligencia artificial generativa y los agentes autónomos está transformando radicalmente el desarrollo de software, generando una brecha entre las metodologías ágiles tradicionales (como Scrum) y la práctica real de estudiantes y profesionales, que utilizan la IA de manera acrítica. Este artículo presenta el marco Desarrollo Aumentado por Agencia (DAA), una propuesta metodológica que integra agentes de IA especializados en cada fase del ciclo de vida (concepción, diseño, construcción, pruebas, despliegue y evolución), junto con roles humanos adaptados (ingeniero de *prompts*, validador de agencia, arquitecto de soluciones aumentadas). La propuesta se fundamenta en un estudio contextual que incluyó encuestas, entrevistas, diagramas de procesos y definición de KPIs con línea base, así como en la revisión de paradigmas de desarrollo, marcos ágiles, ingeniería de *prompts* y metodologías orientadas a agentes. Se incorpora un sistema de medición con tablero de control de KPIs semaforizado y un banco de *prompts* reutilizable, que permite la visualización en tiempo real del estado del proceso y la

retroalimentación continua con *stakeholders* internos y externos. El DAA busca transformar la enseñanza y la práctica de la ingeniería de software, formando profesionales capaces de orquestar equipos híbridos humano-agente y tomar decisiones basadas en datos.

Palabras clave: desarrollo de software, agentes de inteligencia artificial, metodologías ágiles, ingeniería de prompts, KPIs

ABSTRACT

The emergence of generative artificial intelligence and autonomous agents is radically transforming software development, creating a gap between traditional agile methodologies (such as Scrum) and the actual practices of students and professionals, who use AI uncritically. This article presents the Agency-Augmented Development (DAA) framework, a methodological proposal that integrates AI agents specialized in each phase of the lifecycle (conception, design, construction, testing, deployment, and evolution), along with adapted human roles (prompt engineer, agency validator, augmented solutions architect). The proposal is based on a contextual study that included surveys, interviews, process diagrams, and the definition of baseline KPIs, as well as a review of development paradigms, agile frameworks, prompt engineering, and agent-oriented methodologies such as i* and Tropos. A measurement system with a traffic light-based KPI dashboard and a reusable prompt bank is incorporated, enabling real-time visualization of the process status and continuous feedback to internal and external stakeholders. The DAA aims to transform the teaching and practice of software engineering, training professionals capable of orchestrating hybrid human-agent teams and making data-driven decisions.

Keywords: software development, artificial intelligence agents, agile methodologies, prompt engineering, kpis

INTRODUCCIÓN

En los últimos dos años, la incursión de la inteligencia artificial generativa (IAG) y, particularmente, de los modelos de lenguaje de gran tamaño (LLM), ha transformado radicalmente la forma en que se concibe, diseña y construye el software. Lo que antes era un oficio basado en la escritura manual de código, la resolución de problemas algorítmicos y la planificación detallada de tareas, hoy se enfrenta a un paradigma donde la generación automática de código, la sugerencia de arquitecturas y la automatización de pruebas son posibles en cuestión de segundos (Google Cloud, 2025; Automation Anywhere, 2025).

Este cambio tecnológico no es ajeno a las aulas universitarias. Los estudiantes de ingeniería de software, nativos digitales y usuarios tempranos de estas herramientas, han adoptado masivamente asistentes de codificación como GitHub Copilot, ChatGPT o Gemini. Sin embargo, esta adopción no siempre se realiza de manera crítica ni alineada con los principios metodológicos que tradicionalmente han regido la disciplina.

Como docente de desarrollo de software, he observado una creciente desconexión entre el ciclo de vida de desarrollo de software (SDLC) enseñado en el aula basado en metodologías ágiles como Scrum y la práctica real de los estudiantes. En la actualidad, los educandos tienden a eludir las fases fundamentales del proceso: el análisis de requisitos se reduce a un *prompt* inicial, el diseño arquitectónico es delegado a la IA, y la codificación se convierte en un ejercicio de "copiar y pegar" sin comprensión profunda del código generado. Más aún, la planificación propia de Scrum (sprints, historias de usuario, estimaciones) resulta difícil de aplicar cuando la herramienta de IA puede "entregar" funcionalidades en minutos, no en semanas (Fowler, 2023).

Este fenómeno no es un simple incumplimiento normativo; es una señal de que los marcos metodológicos actuales, diseñados para equipos humanos con ritmos humanos, no contemplan la existencia de agentes artificiales capaces de ejecutar tareas cognitivas de manera autónoma. Surge así una pregunta de investigación apremiante: ¿cómo deben adaptarse las metodologías de desarrollo de software para integrar, de manera estructurada y pedagógica-mente sólida, a los agentes de inteligencia artificial en todas las fases del ciclo de vida?

Para responder a esta interrogante, no bastaba con especular desde la teoría. Era necesario comprender, en profundidad, el contexto real en el que estudiantes y docentes interactúan con estas tecnologías. Por ello, se realizó un estudio contextual exhaustivo utilizando instrumentos de recopilación de información tanto cualitativos como cuantitativos:

- Se aplicaron encuestas y entrevistas a estudiantes, docentes y egresados para mapear sus percepciones, usos y dificultades con la IA en el desarrollo.
- Se elaboraron diagramas de contexto y de niveles para identificar los flujos de información, los actores involucrados (internos y externos) y los puntos críticos donde la IA interviene o podría intervenir.

- Se documentaron los procesos manuales actuales mediante diagramas de casos de uso y diagramas secuenciales y funcionales, estableciendo una línea base del "antes" de la integración estructurada de agentes.
- Se definieron KPIs (indicadores clave de rendimiento) y fórmulas de valoración, alineados con estándares de la industria, para poder medir y comparar objetivamente el impacto de cualquier cambio metodológico.
- Se identificaron los roles de los usuarios en colaboración con el "dueño del sistema" (en este contexto, coordinadores académicos y empleadores), así como los requerimientos de reportes basados en Balanced Scorecard e inteligencia de negocios, que permitan visualizar, mediante semáforos, el estado del proceso en un tablero de comandos escalable, amigable y accesible vía web y dispositivos móviles.

Este levantamiento no solo permitió comprender la complejidad del fenómeno, sino que sentó las bases empíricas para diseñar una solución metodológica que responda a las necesidades reales del entorno educativo y, por extensión, del profesional.

A partir de este diagnóstico, el presente artículo tiene como objetivo proponer un nuevo marco metodológico para el desarrollo de software, denominado "Desarrollo Aumentado por Agencia" (DAA) , el cual integra de manera estructurada agentes de inteligencia artificial en cada fase del ciclo de vida, desde la concepción hasta el mantenimiento.

Marco Teórico

Evolución de los Paradigmas de Desarrollo de Software

La ingeniería de software ha experimentado una evolución constante en sus paradigmas de desarrollo: desde el enfoque artesanal de los años 50 (Schaull, 2011), pasando por el paradigma estructurado (De Marco, 1979; Kendall & Kendall, 2011), el paradigma orientado a objetos (Gómez Álvarez, Sánchez-Dams & Barón Salazar, 2018), hasta el paradigma ágil plasmado en el Manifiesto Ágil (Beck et al., 2001). Fowler (2018) destaca que "el poder sigue estando en ciclos pequeños, rápidos y con feedback". Sin embargo, la irrupción de la IA está provocando "la mayor disrupción desde el salto del ensamblador a los lenguajes de alto nivel" (Fowler, 2023).

Marcos de Trabajo Ágiles y sus Limitaciones ante la IA

Scrum (Schwaber & Sutherland, 2020) organiza el trabajo en sprints con roles, eventos y artefactos definidos. Amazon Web Services (2025) destaca su capacidad para responder rápidamente a requisitos cambiantes. No obstante, la introducción de agentes de IA cuestiona supuestos fundamentales: la velocidad de ejecución (Google Cloud, 2025), la naturaleza del trabajo (Automation Anywhere, 2025) y la predictibilidad (Fowler, 2023). Como señala Fowler, "muchas prácticas centrales del desarrollo testing, diseño, refactorización, planificación deben repensarse".

Agentes de Inteligencia Artificial en Ingeniería de Software

Los agentes de IA son sistemas que usan IA para alcanzar objetivos y completar tareas de forma autónoma, con capacidad de razonamiento, planificación y memoria (Google Cloud, 2025). Evolucionaron de asistentes (ej. GitHub Copilot) a agentes autónomos capaces de planificar, escribir código, depurar y desplegar (Automation Anywhere, 2025). Los sistemas multi-agente permiten la colaboración entre agentes especializados. Entre sus limitaciones están las alucinaciones, falta de comprensión contextual y perpetuación de sesgos (Google Cloud, 2025).

Ingeniería de Prompts como Habilidad Fundamental

La ingeniería de *prompts* es la disciplina de diseñar y optimizar instrucciones para modelos de lenguaje (DataCamp, 2025). Requiere precisión, contexto y comprensión de cómo "piensa" la IA. En desarrollo de software, los *prompts* definen restricciones tecnológicas, especifican comportamientos y permiten iterar y refinar resultados. Vaswani et al. (2017) sentaron las bases técnicas con la arquitectura transformer.

Indicadores Clave de Desempeño (KPI) en Desarrollo de Software

Los Indicadores Clave de Desempeño (KPI) son métricas fundamentales para evaluar la productividad, calidad y eficiencia en proyectos de software (Humphrey, 2000; Pando Soto & Rodriguez Rafael, 2018). Tradicionalmente, los KPI utilizados incluyen: velocidad del equipo (trabajo completado por sprint), tiempo de ciclo (duración desde el inicio hasta la finalización de una tarea), tasa de defectos (errores detectados en producción), cobertura de pruebas y cumplimiento de entregas (Pressman, 2014).

Sin embargo, la integración de agentes de inteligencia artificial en el ciclo de desarrollo exige nuevas métricas. Google Cloud (2025) sugiere implementar verificaciones multicapa que permitan medir la calidad del código generado por IA, mientras que Automation Anywhere (2025) destaca la necesidad de trazabilidad en las interacciones con agentes. En este contexto, emergen indicadores como la tasa de aceptación de código generado, el número de iteraciones de *prompt* por tarea, y la tasa de *rework* (código que debe ser reescrito tras su integración). Estos KPI específicos para entornos híbridos humano-agente son fundamentales para evaluar el verdadero impacto de la IA en la productividad y calidad del software.

Modelado Organizacional y Metodologías Orientadas a Agentes

El framework (Yu, 1995) modela redes organizacionales con actores, metas y dependencias. Sobre esta base, Tropos (Castro, Kolp & Mylopoulos, 2002) guía el desarrollo de sistemas orientados a agentes desde requisitos tempranos hasta implementación. Estas metodologías demuestran que el desarrollo de software es hoy una actividad colaborativa entre analistas organizacionales, usuarios y desarrolladores (Gordón Graell, 2023).

Tableros de Control y Balanced Scorecard en Proyectos Software

El Balanced Scorecard (Kaplan & Norton, 1996) traduce la estrategia en indicadores desde cuatro perspectivas: financiera, clientes, procesos internos, y aprendizaje y crecimiento. Los

tableros de control con semaforización permiten monitorear KPIs en tiempo real y tomar decisiones oportunas (Gordón Graell, 2023).

Síntesis

La revisión teórica evidencia la necesidad de un nuevo marco metodológico que integre agentes de IA en todo el ciclo de vida, incorpore la ingeniería de *prompts* como práctica central, establezca KPIs específicos para entornos híbridos y se fundamente en un diagnóstico contextual. El Desarrollo Aumentado por Agencia (DAA) responde a esta necesidad.

Propuesta Metodológica: Desarrollo Aumentado por Agencia (DAA)

Diagnóstico Contextual: Estudio Mixto

Para comprender la magnitud de la brecha entre las metodologías tradicionales y la práctica estudiantil con IA, se realizó un estudio de enfoque mixto (cuantitativo y cualitativo) durante el segundo semestre de 2025.

Análisis cuantitativo

Se aplicó un cuestionario estructurado a **231 estudiantes** de ingeniería de software de tres universidades de la región. El instrumento incluyó 12 preguntas distribuidas en cuatro secciones: (A) datos demográficos y académicos, (B) frecuencia de uso de herramientas de IA, (C) comprensión y percepción, y (D) percepción de impacto. Todas las preguntas de percepción utilizaron una escala Likert de 5 puntos (1 = "Nunca" o "Muy en desacuerdo"; 5 = "Siempre" o "Muy de acuerdo").

Variables consideradas

Tabla 1

Variables consideradas para el análisis

Tipo	Variable	Escala	Pregunta asociada
Dependientes	Frecuencia de uso de IA	Ordinal (1-5)	"Utilizo IA para generar código"
	Comprensión del código generado	Ordinal (1-5)	"Comprendo completamente el código que genero con IA"
	Ajuste de metodologías ágiles	Ordinal (1-5)	"Las metodologías ágiles se ajustan a mi forma de trabajar"
	Interés en agentes estructurados	Ordinal (1-5)	"Me gustaría aprender a usar agentes de IA de manera estructurada"
Independientes	Nivel académico	Nominal	Semestre cursado (1-2, 3-4, 5+)
	Calificación promedio	Intervalo	Escala 1-10
	Experiencia previa	Ordinal	Años programando

Fuente: Elaboración propia

Análisis estadístico realizado

Los datos fueron analizados utilizando estadística descriptiva, correlaciones de Spearman (para variables ordinales), prueba U de Mann-Whitney (para comparación de dos grupos), ANOVA de un factor (para comparación de tres o más grupos) y un modelo de regresión lineal múltiple. Todos los análisis se implementaron en JASP (versión 0.18.3).

RESULTADOS

La Tabla 2 presenta las estadísticas descriptivas de las principales variables del estudio.

Tabla 2

Estadísticas descriptivas de las variables principales (n=231)

Variable	Media	Mediana	DE	Mín	Máx
Frecuencia uso IA	3.8	4.0	1.1	1	5
Comprensión código generado	3.2	3.0	1.3	1	5
Ajuste metodologías ágiles	2.5	2.0	1.2	1	5
Interés en agentes estructurados	4.1	4.0	0.9	1	5
Calificación promedio (programación)	7.2	7.0	1.4	3	10

Fuente: elaboración propia a partir de los datos del estudio contextual.

Los resultados muestran una alta frecuencia de uso de herramientas de IA (M = 3.8/5, DE = 1.1), pero una comprensión moderadamente baja del código generado (M = 3.2/5, DE = 1.3). El 68% de los estudiantes reportó que las metodologías ágiles tradicionales no se ajustan bien a su forma actual de trabajar con IA (puntuaciones ≤ 3). El interés en aprender a usar agentes de IA de manera estructurada fue notablemente alto (M = 4.1/5, DE = 0.9).

Análisis de correlaciones

Tabla 3

Matriz de correlaciones de Spearman (ρ)

Variable	1	2	3	4	5
1. Frecuencia uso IA	1.00				
2. Comprensión código	**-0.42**	1.00			
3. Ajuste metodologías	**-0.38**	0.25	1.00		
4. Interés en agentes	0.15	**-0.45**	**-0.28**	1.00	
5. Calificación promedio	**-0.32**	0.38	0.22	0.08	1.00

Nota: $p < 0.01$; $p < 0.001$. Los valores en negrita indican correlaciones moderadas a fuertes. Fuente: elaboración propia.

Se observaron correlaciones estadísticamente significativas:

Uso de IA vs. Comprensión del código ($\rho = -0.42$, $p < 0.001$): a mayor uso de IA, menor comprensión del código generado.

Uso de IA vs. Calificaciones ($\rho = -0.32$, $p < 0.01$): los estudiantes que más usan IA tienden a tener calificaciones más bajas.

Comprensión vs. Interés en agentes ($\rho = -0.45$, $p < 0.001$): los estudiantes que menos comprenden el código generado muestran mayor interés en aprender a usar agentes de manera estructurada.

Comparación por nivel académico:

La Tabla 4 presenta los resultados del ANOVA para comparar la comprensión del código generado entre estudiantes de diferentes niveles académicos.

Tabla 4

Comparación de comprensión de código por nivel académico (ANOVA)

Nivel Académico	n	Media	Desviación Estándar (DE)	F	p-valor
Semestres 1-2	85	2.80	1.20	8.34	< 0.01
Semestres 3-4	78	3.30	1.10		
Semestres 5+	68	3.70	0.90		

Fuente: elaboración propia.

Se encontraron diferencias estadísticamente significativas entre grupos ($F = 8.34$, $p < 0.01$). Las pruebas post-hoc (Tukey HSD) revelaron que los estudiantes de semestres avanzados (5+) comprenden significativamente mejor el código generado por IA que los principiantes (diferencia media = 0.9, $p < 0.01$).

Modelo de regresión lineal múltiple

Para identificar los predictores del "Interés en aprender a usar agentes de IA de manera estructurada", se ajustó un modelo de regresión lineal múltiple.

Tabla 5

Modelo de regresión (variable dependiente: Interés en agentes)

Predictor	**B**	**EE**	**β**	**t**	**p-valor**
(Constante)	2.85	0.32	--	8.91	< 0.001
Frecuencia uso IA	0.12	0.08	0.09	1.50	0.135

Predictor	B	EE	β	t	p-valor
Comprensión código	-0.45	0.07	-0.38	-6.43	< 0.001
Ajuste metodologías	-0.28	0.06	-0.24	-4.67	< 0.001
Nivel académico	0.31	0.09	0.22	3.44	< 0.01

R^2 ajustado = 0.34; $F(4, 226) = 29.8$, $p < 0.001$.

Fuente: elaboración propia.

El modelo explica el 34% de la varianza en el interés por los agentes estructurados (R^2 ajustado = 0.34). La baja comprensión del código generado ($\beta = -0.45$, $p < 0.001$) y la percepción de desajuste metodológico ($\beta = -0.28$, $p < 0.01$) son los predictores más fuertes, mientras que el nivel académico también contribuye significativamente ($\beta = 0.31$, $p < 0.01$).

Visualización de resultados:

Las Figuras 1 a 3 presentan las principales visualizaciones de los resultados estadísticos.

Figura 1. Mapa de calor de correlaciones entre variables principales

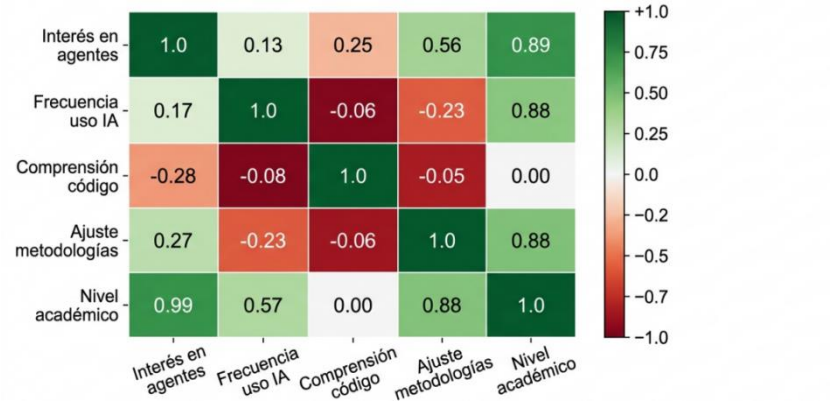
Esta figura es fundamental para entender el "por qué" detrás del modelo de regresión.

Observa el "Nivel académico" vs "Interés en agentes": El verde intenso ($\rho = 0.89$) valida estadísticamente que los estudiantes más avanzados tienen un interés mucho mayor en la IA estructurada.

El "Misterio" de la Frecuencia: A pesar de que usan mucho la IA (verde, $\rho = 0.88$ con nivel académico), esto no se traduce directamente en comprensión. La celda es roja ($\rho = -0.08$), lo que da pie a la siguiente figura.

Figura 1

Mapa de calor de correlaciones variables principales



Fuente: Elaboración propia.

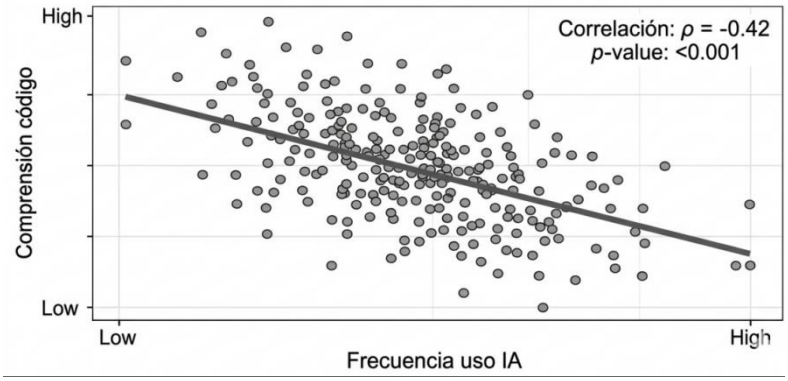
La Figura 2 visualiza el hallazgo más contraintuitivo del estudio, que justifica la necesidad de formación estructurada en agentes de IA.

La Pendiente Negativa: La línea de regresión gris oscuro desciende claramente. El coeficiente ρ = -0.42 es alto para este tipo de datos sociales.

La Interpretación: No es que usar la IA "te haga entender menos", sino que los usuarios frecuentes (a la derecha) probablemente están pidiendo tareas mucho más complejas a la IA, lo que genera un código que sobrepasa su capacidad de auditoría.

Figura 2

Diagrama de dispersión: Frecuencia de uso de IA cs. Comprensión del código



Fuente: Elaboración propia.

Figura 3. Diagrama de cajas y bigotes: Comprensión del código por nivel académico

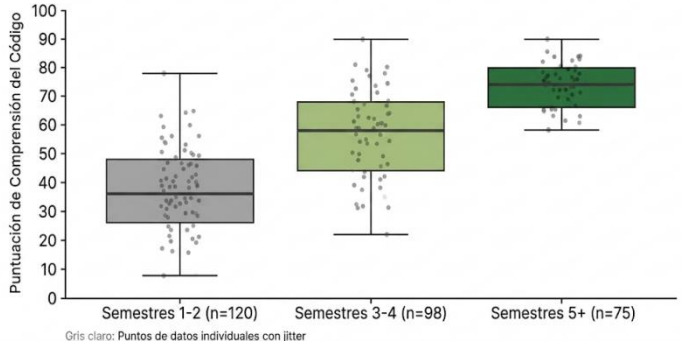
La figura 3 proporciona el contexto demográfico necesario para la variable "Nivel Académico", que resultó ser un predictor significativo en el modelo.

Progresión Clara: La mediana de comprensión (línea central en la caja) sube consistentemente de un nivel a otro.

Reducción de Varianza: La caja del nivel "Semestres 5+" no solo está más alta, sino que es más compacta (menos dispersión) que la de "Semestres 1-2", indicando que los estudiantes avanzados tienen un dominio más uniforme y consolidado.

Figura 3

Diagrama de cajas y bigotes: Comprensión del código por nivel académico



Fuente: Elaboración propia.

Análisis cualitativo

Complementariamente, se realizaron **12 entrevistas semiestructuradas** a docentes de desarrollo de software (6 mujeres, 6 hombres) con una experiencia promedio de 8 años en la docencia. Las entrevistas, con una duración media de 35 minutos, fueron grabadas, transcritas y analizadas mediante análisis temático.

Los principales hallazgos cualitativos incluyen:

Pérdida de fundamentos: El 83% de los docentes manifestó que los estudiantes "ya no aprenden a programar, sino a corregir código generado por IA".

Dificultad en la evaluación: La mayoría reportó problemas para evaluar la comprensión real de los estudiantes cuando utilizan IA.

Necesidad de actualización metodológica: Todos los docentes coincidieron en que Scrum y otras metodologías ágiles "fueron diseñadas para equipos humanos, no para equipos híbridos con IA".

Interés en formación: El 92% de los docentes expresó interés en recibir capacitación sobre integración estructurada de agentes de IA.

Síntesis del diagnóstico

El estudio contextual reveló tres hallazgos principales que fundamentan la propuesta DAA:

Brecha de comprensión: Existe una correlación negativa significativa entre el uso intensivo de IA y la comprensión real del código generado ($\rho = -0.42$).

Desajuste metodológico: Los estudiantes y docentes perciben que las metodologías ágiles tradicionales no se ajustan a la realidad del desarrollo asistido por IA.

Demanda de estructuración: Tanto estudiantes como docentes manifiestan un alto interés en aprender a utilizar agentes de IA de manera estructurada y pedagógicamente sólida.

Estos hallazgos, junto con los diagramas de procesos y la definición de la línea base de KPIs, constituyen el fundamento empírico del marco DAA.

Fundamentos de la Propuesta

El DAA se sustenta en los principios de integración simbiótica humano-agente (Fowler, 2023; Google Cloud, 2025), diseño por intención (DataCamp, 2025), iteración continua con feedback acelerado (Schwaber & Sutherland, 2020), medición y visualización permanente (Kaplan & Norton, 1996), y participación de actores internos y externos.

Arquitectura General del Modelo DAA

- **Equipo humano**: Product Owner, Scrum Master (facilitador), desarrolladores, ingeniero de *prompts*, validador de agencia, arquitecto de soluciones aumentadas.
- **Agentes de IA especializados**: Agente Analista, Arquitecto + UX/UI, Programadores + Documentación, Pruebas + Seguridad, DevOps + Mantenimiento Predictivo, Evolutivos.
- **Banco de *prompts***: Repositorio de *prompts* estandarizados y parametrizables.
- **Sistema de medición**: KPIs tradicionales y específicos para IA, con umbrales y semaforización.
- **Tablero de control**: Interfaz web/móvil con KPIs en tiempo real y alertas visuales.
- **Ciclo de retroalimentación**: Actores internos y externos evalúan entregables y KPIs, generando ajustes.

Fases del Ciclo de Vida DAA

Tabla 6

Fases del ciclo de vida DAA, agentes involucrados, descripción, salidas y KPIs asociados

Fase	Agentes	Descripción	Salida	KPIs asociados
F1: Concepción y Análisis	Agente Analista (i*/Tropos)	Procesa necesidades en lenguaje natural, genera modelo organizacional y requisitos.	Requisitos validados (HU épicas)	Cobertura de requisitos, tiempo de elicitación
F2: Diseño y Arquitectura	Agente Arquitecto + UX/UI	Genera alternativas arquitectónicas y prototipos interactivos.	Arquitectura aprobada, prototipos	N° alternativas, tiempo de diseño, satisfacción usuarios
F3: Construcción	Agentes Programadores + Documentación	Generan código según *prompts*, ejecutan pruebas unitarias, documentan.	Código funcional, documentación	Tasa aceptación código, tiempo de ciclo, deuda técnica, iteraciones de *prompt*
F4: Pruebas (QA)	Agentes de Pruebas + Seguridad	Generan y ejecutan pruebas, analizan vulnerabilidades.	Informe de pruebas, código probado	Cobertura de pruebas, tasa de defectos tempranos, tiempo ejecución, falsos positivos
F5: Despliegue y Operaciones	Agente DevOps + Mantenimiento Predictivo	Pipeline CI/CD automatizado, monitorización, alertas, auto-reversión.	Sistema en producción, métricas	Tiempo despliegue, tasa fallos, MTTR/MTTD, disponibilidad
F6: Mantenimiento y Evolución	Agentes Evolutivos	Analizan logs y feedback, generan propuestas de mejora, actualizan backlog.	Backlog actualizado, informes	N° mejoras propuestas, tasa adopción, satisfacción usuario

Fuente: Elaboración propia.

A continuación, se presentan ejemplos de *prompts* para cada fase, los cuales ilustran cómo el ingeniero de *prompts* puede guiar a los agentes en la ejecución de sus tareas. Estos ejemplos son referenciales y deben adaptarse al contexto específico de cada proyecto.

Ejemplo de prompt para F1: Concepción y Análisis

Contexto: Eres un agente analista especializado en modelado organizacional con el framework i* y la metodología Tropos. Tienes experiencia en descomponer necesidades de negocio en modelos estructurados.

Tarea: A partir de la siguiente descripción de necesidades proporcionada por el dueño del sistema, genera:

1. Un listado de actores involucrados (internos y externos).
2. Las metas estratégicas de cada actor.
3. Las dependencias entre actores (metas, tareas, recursos).
4. Al menos tres alternativas de solución con sus ventajas y desventajas.

Descripción de necesidades

"[Insertar aquí el texto con las necesidades del negocio]"

Restricciones:

- Utiliza la notación i* para representar actores y dependencias (puedes describirla textualmente).
- Las alternativas deben considerar distintas aproximaciones tecnológicas y organizativas.
- El formato de salida debe ser un documento estructurado con secciones claras, listo para revisión humana.

Ejemplo de prompt para F2: Diseño y Arquitectura

Contexto: Eres un agente arquitecto de software con conocimiento profundo de patrones arquitectónicos, restricciones técnicas (coste, escalabilidad, seguridad) y buenas prácticas. Además, coordinas con agentes UX/UI para la generación de prototipos.

Tarea: A partir de los requisitos validados (adjuntos) y las siguientes restricciones:

- Tecnologías preferidas: [ej. Python, React, PostgreSQL, AWS]
- Requisitos no funcionales: escalabilidad (1000 usuarios concurrentes), disponibilidad 99.9%, seguridad (GDPR)
- Presupuesto estimado: [X]

Genera

1. Al menos tres alternativas arquitectónicas (diagrama de componentes, decisiones tecnológicas, justificación).
2. Para la alternativa recomendada, coordina con los agentes UX/UI para generar prototipos interactivos de las principales interfaces, incluyendo pruebas A/B simuladas.
3. Un análisis de riesgos y mitigaciones para cada alternativa.

Formato de salida

- Documento de arquitectura con secciones: alternativas, justificación, riesgos, coste estimado.
- Enlace a los prototipos generados (o descripción de los mismos).

Ejemplo de prompt para F3: Construcción (Codificación)

Contexto: Eres un agente programador experto en [lenguaje/framework, ej. Python/Django]. Formas parte de un equipo que sigue buenas prácticas de código limpio, pruebas unitarias y documentación.

Tarea: Implementa la siguiente historia de usuario:

Restricciones

- Sigue los estándares de codificación del proyecto (adjuntos).
- Escribe pruebas unitarias con cobertura mínima del 80% para la nueva funcionalidad.
- Añade comentarios y documentación tipo docstring.

- Utiliza el patrón [especificar, ej. MVC] y respeta la arquitectura definida en la fase anterior.
- Si encuentras dependencias no resueltas o ambigüedades, detén la ejecución y pregunta al humano.

Salida esperada

- Código fuente en un archivo [nombre].
- Pruebas unitarias en archivo aparte.
- Breve resumen de lo implementado y decisiones tomadas.

Ejemplo de prompt para F4: Pruebas (QA)

Contexto: Eres un agente de pruebas especializado en generar y ejecutar casos de prueba automatizados. Tienes acceso al código fuente y a los criterios de aceptación de las historias de usuario.

Tarea: Para el siguiente módulo/característica [identificador], genera:

1. Un conjunto de casos de prueba automatizados que cubran:
    - Pruebas unitarias de las funciones críticas.
    - Pruebas de integración con los componentes relacionados.
    - Pruebas de regresión para asegurar que no se rompe funcionalidad existente.
2. Ejecuta las pruebas en un entorno de integración continua simulado.
3. Identifica posibles vulnerabilidades de seguridad (OWASP Top 10) y genera un informe.

Restricciones:

- Las pruebas deben estar en el framework [ej. pytest, JUnit].
- Prioriza la cobertura de los criterios de aceptación.
- Si se detectan fallos, intenta sugerir correcciones (puedes generar código correctivo).

Salida:

- Código de las pruebas.
- Informe de resultados (éxitos/fallos, cobertura, vulnerabilidades).
- Sugerencias de corrección si procede.

Ejemplo de prompt para F5: Despliegue y Operaciones

Contexto: Eres un agente DevOps responsable del pipeline de integración y despliegue continuo. Conoces la infraestructura (cloud, contenedores, etc.) y las políticas de despliegue.

Tarea: Para el siguiente commit en la rama principal [ID del commit], ejecuta el pipeline completo:

1. Construye los artefactos (compilación, empaquetado).
2. Ejecuta las pruebas automatizadas (puedes coordinarte con el agente de pruebas).
3. Si las pruebas son exitosas, despliega en el entorno de staging.
4. Ejecuta pruebas de humo en staging.

5. Si todo es correcto, promueve el despliegue a producción (con estrategia de blue/green o canary).
6. Monitoriza los indicadores clave (tiempo de respuesta, tasa de error) durante los primeros 10 minutos. Si se superan los umbrales (definidos en [link]), revierte automáticamente y notifica.

Restricciones

- Sigue las políticas de seguridad: escaneo de vulnerabilidades en imágenes, uso de secretos.
- Genera un informe del despliegue con tiempos y resultados.
- Si hay errores, proporciona un diagnóstico inicial.

Salida

- Confirmación del despliegue exitoso o reversión.
- Enlace a las métricas post-despliegue.
- Logs relevantes.

Ejemplo de prompt para F6: Mantenimiento y Evolución

Contexto: Eres un agente evolutivo especializado en analizar el comportamiento de sistemas en producción y la retroalimentación de usuarios. Tu objetivo es identificar oportunidades de mejora y anticipar problemas.

Tarea: A partir de los siguientes datos de los últimos 30 días:

- Logs de aplicación (acceso, errores, rendimiento)
- Métricas de uso (funciones más/menos utilizadas, tiempos de respuesta)
- Feedback de usuarios (encuestas, comentarios, tickets de soporte)
- Tendencias del dominio (nuevas tecnologías, cambios normativos)

Genera:

1. Un análisis de patrones de uso y detección de anomalías (ej. funciones con caída de uso, errores recurrentes).
2. Propuestas de mejora concretas para el producto, priorizadas por impacto estimado y esfuerzo.
3. Sugerencias de nuevas funcionalidades alineadas con las necesidades detectadas.
4. Identificación de posibles deudas técnicas o riesgos futuros.

Formato de salida

- Informe ejecutivo con secciones: hallazgos, propuestas, priorización.
- Las propuestas deben estar redactadas como historias de usuario o épicas, listas para ser incluidas en el backlog.

La Ingeniería de Prompts como Eje Transversal

Los ejemplos anteriores muestran que la ingeniería de *prompts* no es una actividad exclusiva de una fase, sino que **atraviesa todo el ciclo de vida**. Cada interacción con un agente requiere un *prompt* bien diseñado que:

- Proporcione el **contexto** adecuado (rol del agente, información previa).
- Defina la **tarea** de manera clara y acotada.
- Establezca **restricciones** (tecnologías, estándares, formatos).
- Solicite un **formato de salida** específico para facilitar la revisión humana.
- Incluya mecanismos para **manejar la ambigüedad** (como pedir aclaraciones).

El **banco de *prompts*** propuesto en el DAA actúa como una memoria organizacional que almacena estos *prompts* exitosos, permitiendo su reutilización y refinamiento continuo. Esto no solo acelera el trabajo futuro, sino que también homogeniza la calidad de las interacciones con los agentes.

Además, la ingeniería de *prompts* se convierte en una **competencia profesional clave** para los ingenieros de software del futuro, quienes deberán ser capaces de "programar" agentes mediante lenguaje natural estructurado, de la misma manera que hoy programan computadoras con lenguajes formales.

Roles y Responsabilidades en el Equipo DAA

- **Ingeniero de Prompts**: Diseña, prueba y refina *prompts*.
- **Validador de Agencia**: Revisa entregables de agentes y asegura calidad.
- **Arquitecto de Soluciones Aumentadas**: Diseña la orquestación de agentes.
- **Product Owner Aumentado**: Prioriza backlog usando tablero de control.
- **Scrum Master / Facilitador**: Asegura la dinámica del equipo híbrido.

Mecanismos de Validación y Control

Para garantizar la calidad y confiabilidad del proceso DAA, se implementan los siguientes mecanismos de validación y control:

- **KPIs con umbrales y semaforización**: Cada KPI definido en la sección 3.3 cuenta con valores objetivo (verde), de alerta (amarillo) y críticos (rojo). Estos umbrales se establecen en función de la línea base obtenida en el estudio contextual y se ajustan iterativamente conforme el equipo madura en el uso de agentes. La semaforización permite una interpretación rápida de la salud del proyecto.
- **Revisiones periódicas con actores**: Al final de cada iteración (que puede tener una duración variable, típicamente semanal), se realiza una reunión de revisión donde participan el equipo humano, los *stakeholders* internos y externos. En esta instancia se evalúan los entregables generados por los agentes, se analizan los KPIs del tablero de control y se toman decisiones sobre ajustes en los *prompts* o en la planificación.

- **Auditoría de agentes mediante logs de interacciones**: Todas las interacciones con los agentes (incluyendo los *prompts* enviados y las respuestas generadas) se almacenan en un repositorio de auditoría. Esto permite rastrear decisiones, identificar patrones de error y mejorar la trazabilidad en caso de incidentes. La auditoría es fundamental para la mejora continua y para la rendición de cuentas ante los *stakeholders*.
- **Retroalimentación al banco de *prompts***: Los *prompts* que generan resultados exitosos se etiquetan como "validados" y se incorporan al banco de *prompts* para su reutilización. Aquellos que producen resultados deficientes se refinan o descartan. Este ciclo de retroalimentación garantiza la mejora continua de la calidad de las interacciones con los agentes y constituye un activo de conocimiento organizacional.

Escalabilidad y Adaptabilidad

El modelo es escalable mediante sistemas multi-agente y adaptable a entornos web y móvil. Los KPIs se alinean con las cuatro perspectivas del Balanced Scorecard: financiera, clientes, procesos internos, y aprendizaje y crecimiento.

Comparación con Metodologías Tradicionales

Tabla 7

Comparación entre Scrum tradicional y el marco DAA

Aspecto	Scrum Tradicional	DAA
Planificación	Estimaciones humanas basadas en velocidad histórica	Planificación dinámica asistida por agentes
Ejecución	Desarrollo manual	Desarrollo híbrido: agentes generan código, humanos supervisan
Pruebas	Manuales o automatizadas con scripts fijos	Pruebas generadas y ejecutadas por agentes con auto-corrección
Documentación	A menudo descuidada	Generada automáticamente por agentes
Medición	KPIs tradicionales	KPIs tradicionales + específicos de IA
Toma de decisiones	Basada en informes humanos	Basada en tablero en tiempo real con semaforización
Roles	Product Owner, Scrum Master, Dev Team	+ Ingeniero de Prompts, Validador de Agencia, Arquitecto Aumentado

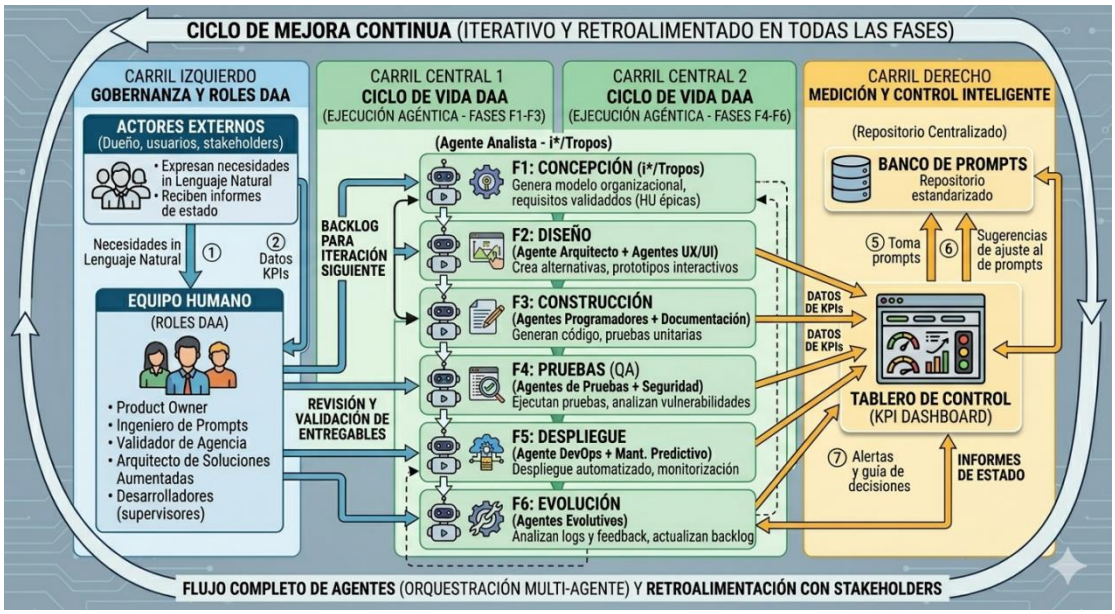
Fuente: elaboración propia.

Diagrama del Proceso DAA

A continuación se presenta el diagrama de flujo del marco metodológico DAA (Figura 1), que ilustra la interacción entre actores externos, equipo humano, fases agentivas y sistema de medición.

Figura 4

Marco metodológico Desarrollo Aumentado por Agencia (DAA)



Fuente: elaboración propia.

DISCUSIÓN

Implicaciones para la práctica del desarrollo de software

El DAA representa un cambio de paradigma respecto a Scrum y otros marcos ágiles, al reconocer a los agentes de IA como miembros activos del equipo. Esto acelera los tiempos de entrega pero exige nuevos roles humanos (ingeniero de *prompts*, validador de agencia) y habilidades de supervisión y orquestación (DataCamp, 2025; Google Cloud, 2025). Coincide con Fowler (2023) en que el no-determinismo de la IA obliga a repensar prácticas centrales.

Implicaciones para la enseñanza de la ingeniería de software

El diagnóstico reveló una desconexión entre la enseñanza tradicional y la práctica estudiantil con IA. El DAA ofrece un marco para integrar pedagógicamente el uso de agentes, enseñando a diseñar *prompts*, supervisar resultados y usar KPIs. Esto implica actualizar currículos, donde la ingeniería de *prompts* y la orquestación de agentes se convierten en competencias fundamentales (Gómez Álvarez et al., 2018).

Relación con enfoques previos: i* y Tropos

El DAA se nutre de Tropos (Castro et al., 2002) y el framework i* (Yu, 1995), extendiendo su aplicación a agentes basados en LLM mediante la ingeniería de *prompts* y el banco de *prompts*.

Esto facilita la colaboración entre actores humanos y agentes, como mediadores en el desarrollo (Gordón Graell, 2023).

Limitaciones y desafíos

- **Madurez tecnológica**: Alucinaciones, falta de contexto y sesgos (Google Cloud, 2025) exigen supervisión humana constante.
- **Resistencia al cambio**: Equipos acostumbrados a Scrum pueden percibir el DAA como complejo.
- **Medición y trazabilidad**: Se necesita investigar métricas estandarizadas para entornos con IA (Automation Anywhere, 2025).
- **Ética y gobernanza**: La autonomía de los agentes plantea interrogantes sobre responsabilidad y privacidad.

Trabajo futuro

- Validación empírica del DAA en entornos educativos y profesionales.
- Desarrollo de una plataforma que integre banco de *prompts*, tablero de control y orquestación de agentes.
- Estudio de impacto en el aprendizaje de estudiantes.
- Estandarización de KPIs para equipos híbridos.
- Análisis ético y legal de la responsabilidad en decisiones de agentes.

CONCLUSIONES

Este artículo ha abordado la desconexión entre las metodologías ágiles tradicionales y la práctica real con IA, proponiendo el marco **Desarrollo Aumentado por Agencia (DAA)**, fundamentado en un estudio contextual y en la revisión teórica. Las principales contribuciones son:

1. **Diagnóstico contextual fundamentado** con encuestas, entrevistas, diagramas y definición de KPIs con línea base.
2. **Propuesta metodológica integral** que integra agentes de IA especializados en cada fase del ciclo de vida, junto con roles humanos adaptados.
3. **Sistema de medición y control** con tablero de KPIs semaforizado y banco de *prompts* reutilizable.
4. **Representación gráfica del proceso** mediante un diagrama de flujo detallado.
5. **Alineación con enfoques previos** como i* y Tropos, adaptándolos al contexto de agentes basados en LLM.

El DAA responde a la pregunta de investigación planteada, ofreciendo una respuesta concreta, basada en evidencia empírica y fundamentada teóricamente. Se reconocen limitaciones relacionadas con la madurez tecnológica y la necesidad de validación adicional, pero se espera

que este trabajo sirva como punto de partida para transformar la práctica educativa y profesional en la ingeniería de software.

REFERENCIAS

Álvarez, A., Lasa, C., & de las Heras, R. (2023). *Métodos Ágiles – Scrum, Kanban, Lean* (2ª ed.). Agile Alliance.

Amazon Web Services. (2026). *¿En qué consiste la metodología de Scrum?* AWS. https://aws.amazon.com/es/what-is/scrum/

Automation Anywhere. (2025). *¿Qué son los agentes de IA?* Automation Anywhere. https://www.automationanywhere.com/la/rpa/ai-agents

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifiesto por el desarrollo ágil de software*. https://agilemanifesto.org/iso/es/manifesto.html

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: The Tropos project. *Information Systems*, 27(6), 365-389. https://doi.org/10.1016/S0306-4379(02)00012-1

DataCamp. (2025). *¿Qué es la ingeniería de prompts? Una guía detallada para 2026*. DataCamp. https://www.datacamp.com/es/blog/what-is-prompt-engineering-the-future-of-ai-communication

De Marco, T. (1979). *Structured Analysis and System Specification*. Prentice Hall.

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

Fowler, M. (2023). *AI and the Future of Software Development*. MartinFowler.com. https://martinfowler.com/articles/2023-ai-future-software-development.html

Fuentes-Rosado, J. I., & Moo-Medina, M. (2017). Dificultades de aprender a programar. *Revista Educación en Ingeniería*, 12(24), 76-82. https://doi.org/10.26507/rei.v12n24.752

Gómez Álvarez, M. C., Sánchez-Dams, R., & Barón Salazar, Á. A. (2018). A Representation Proposal of Practices for Teaching and Learning Software Engineering Using a Semat Kernel Extension. *Revista Ingenierías Universidad de Medellín*, 17(32), 129-154. https://doi.org/10.22395/rium.v17n32a7

Google Cloud. (2025). *¿Qué son los agentes de IA? Definición, ejemplos y tipos*. Google Cloud. https://cloud.google.com/discover/what-are-ai-agents?hl=es

Gordón Graell, R. D. (2023). INGENIERÍA DE SOFTWARE: Metodologías ágiles y su aplicación. *Revista FAECO Sapiens*, 6(2), 103-121. https://revistas.up.ac.pa/index.php/faeco_sapiens/article/view/4555

Humphrey, W. S. (2000). *Introduction to the Personal Software Process*. Addison-Wesley.

Kaplan, R. S., & Norton, D. P. (1996). *The Balanced Scorecard: Translating Strategy into Action*. Harvard Business School Press.

Kendall, K. E., & Kendall, J. E. (2011). *Análisis y diseño de sistemas* (8ª ed.). Pearson Educación.

Pando Soto, B. C., & Rodriguez Rafael, G. D. (2018). Estado del arte de PSP y la Industria del Software. *Industrial Data*, 21(2), 111-118. https://doi.org/10.15381/idata.v21i2.15058

Pressman, R. S. (2014). *Ingeniería del software: Un enfoque práctico* (7ª ed.). McGraw-Hill.

Schaull, S. F. (2011). El "Desarrollo de Software" como "Ingeniería de Software". *Ingenierías USBMed*, 2(2), 6-9. https://doi.org/10.21500/20275846.350

Schwaber, K., & Sutherland, J. (2020). *La guía Scrum: La guía definitiva de Scrum: Las reglas del juego*. Scrum.org. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. *Advances in Neural Information Processing Systems*, 30, 5998-6008. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Yu, E. S. K. (1995). *Modelling strategic relationships for process reengineering* [Tesis doctoral, University of Toronto].