Capítulo 1. Introducción

1.1 Contexto y panorama general

El diseño de interfaces de usuario (UI) y la experiencia de usuario (UX) se ha consolidado como un eje estratégico en el desarrollo de productos digitales contemporáneos. La masificación de servicios basados en la Web y aplicaciones móviles ha elevado las expectativas de los usuarios respecto de la eficacia, eficiencia, satisfacción y accesibilidad de los sistemas (ISO, 2018; Preece, Sharp, & Rogers, 2015). En paralelo, la adopción extendida de enfoques ágiles ha comprimido los ciclos de diseño–desarrollo, tensionando la incorporación sistemática de prácticas de evaluación temprana de usabilidad (Beck, 2004; Rubin, 2012).

Dentro de este escenario emergen los denominados *usability* o *UX smells*, analogía conceptual de los *code smells* (Garrido, Rossi, & Distante, 2010; Grigera et al., 2017), como indicios de potenciales problemas de interacción, navegación, claridad semántica, consistencia visual o carga cognitiva que, de no atenderse oportunamente, se traducen en deuda de UX (DaFonseca, 2019; Baltes, 2021) y en sobrecostos posteriores de refactorización. Detectar estos indicios en estadios tempranos —particularmente en prototipos de baja y media fidelidad— ofrece una oportunidad para reducir retrabajo y mejorar la transferencia del diseño a la implementación (Sommerville, 2016; Norman, 2013).

En los últimos años se han propuesto enfoques semi-automáticos para la detección de *usability smells* sobre aplicaciones ya desarrolladas (Grigera et al., 2017; Grigera et al., 2019), marcos de *refactoring* para mejorar la experiencia (Garrido et al., 2010) y herramientas de apoyo a pruebas colaborativas (Grigera et al., 2021). Sin embargo, existe todavía una brecha entre esas aproximaciones posteriores al desarrollo y la disponibilidad de soporte automatizado incrustado en el flujo cotidiano de prototipado visual. Esta tesina aborda dicha brecha mediante la definición e implementación de un plugin de Figma orientado a la detección automática (sin intervención manual) de un subconjunto de *usability smells* estructurales.

1.2 Motivación

La motivación central surge de cuatro ejes convergentes:

1. **Costo del descubrimiento tardío**: Los problemas de usabilidad identificados en etapas avanzadas elevan el esfuerzo de corrección y propagan deuda (Cunningham, 1992; Garrido et al., 2010).
2. **Limitaciones del testeo manual aislado**: Las evaluaciones heurísticas o pruebas con usuarios son efectivas pero consumen tiempo y requieren expertos (Nielsen & Molich, 1990; Nielsen & Loranger, 2006).
3. **Opacidad en prototipos Lo-Fi**: Las herramientas de baja fidelidad priorizan velocidad de ideación pero no integran mecanismos nativos de alerta sobre malas prácticas recurrentes.
4. **Necesidad de especificación transferible**: Ambigüedades no resueltas en prototipos generan decisiones ad-hoc en desarrollo, degradando consistencia y alineación (Pressman, 2010; DaSilva, 2011).

La fundamentación de la propuesta resalta, de manera inequívoca, la importancia estratégica de anticipar la identificación de "usability smells" en las fases iniciales del proceso de diseño, previo a la consolidación de prototipos de alta fidelidad o la materialización en código. En consonancia, investigaciones recientes centradas en el análisis del esfuerzo de interacción (Grigera et al., 2019) y en el desarrollo de herramientas colaborativas para pruebas de usabilidad (Grigera et al., 2021) evidencian que la integración de métricas y sistemas instrumentados en el flujo creativo puede optimizar sustancialmente la toma de decisiones en etapas tempranas. Este enfoque no solo contribuye a la reducción de retrabajo y deuda de UX, sino que también fortalece la transferencia efectiva del diseño hacia la implementación, alineando el proceso con estándares internacionales de calidad y eficiencia en productos digitales.

1.3 Problema y preguntas de investigación

A partir del análisis de la literatura y de la propuesta formal, se enuncia el problema principal:

Falta de un mecanismo automatizado, integrado al entorno de prototipado, que detecte de manera temprana y con bajo ruido un conjunto relevante de usability smells en diseños de baja o media fidelidad sin requerir intervención manual experta.

A partir de lo anterior se plantean las siguientes preguntas de investigación (PI):

- **PI1**: ¿Cuáles *usability smells* presentan señales estructurales observables (como tamaño, proximidad, nomenclatura o texto) suficientes para permitir su detección automática en prototipos que no cuentan con lógica ejecutable?
- **PI2**: ¿Qué plataforma de prototipado proporciona las mejores condiciones (API, modelo de objetos, adopción, extensibilidad) para implementar detectores no intrusivos?
- **PI3**: ¿Cuál es el conjunto mínimo viable de heurísticas que logra equilibrar una adecuada cobertura con una baja tasa de falsos positivos en entornos de diseño ágiles?
- **PI4**: ¿De qué manera es posible modelar y documentar los detectores para facilitar su extensión incremental, incorporando nuevos tipos de datos, reglas o métricas?

1.4 Fundamentación teórica del enfoque propuesto

Diversos estudios recientes sugieren que un subconjunto relevante de "usability smells" de carácter estructural—como la detección de campos con tamaño inadecuado, inconsistencias en formularios, empleo de texto libre en entradas que requieren formato restringido o presencia de vínculos ambiguos—puede ser identificado de manera automática mediante el análisis de metadatos de capas, geometrías y contenido textual accesibles a través de la API de Figma. La aplicación de heurísticas calibradas permitiría mantener la tasa de falsos positivos en márgenes aceptables, incrementando la precisión del sistema propuesto.

Desde la perspectiva de la ingeniería de software orientada a la experiencia de persona usuaria, la integración temprana de mecanismos de detección en el punto de diseño, concretamente mediante un plugin para Figma, representa una estrategia eficiente para prevenir la acumulación de deuda de UX. Este enfoque no solo posibilita la aplicación anticipada de micro-refactorizaciones en la interfaz, sino que también habilita un modelo de mejora continua basado en la iteración y el aprendizaje incremental dentro de los flujos ágiles de trabajo. De este modo, la propuesta plantea que la automatización de la detección, sumada a la inmediatez de la retroalimentación, tiene potencial para transformar positivamente las prácticas de diseño al fomentar mayor alineación con estándares de calidad, robustez y eficiencia en productos digitales.

1.5 Objetivo general

Diseñar, implementar y documentar un plugin extensible para Figma que permita la detección automática temprana de un conjunto priorizado de *usability smells* estructurales en prototipos de interfaces, suministrando retroalimentación accionable y exportable para su refactorización.

1.6 Objetivos específicos

1. **OE1**. Analizar comparativamente herramientas de prototipado (Figma, Balsamiq, Adobe XD, Axure, Sketch, InVision, Eve, Azure) bajo criterios de extensibilidad, modelo de representación y soporte comunitario.
2. **OE2**. Construir una taxonomía de *usability smells* candidatos a detección automática sin intervención (señales estructurales observables) y clasificar su viabilidad.
3. **OE3**. Diseñar heurísticas y detectores parametrizables para los smells priorizados (tamaño, consistencia, formato, vínculos confusos, valores limitados, complejidad y datos sensibles).
4. **OE4**. Implementar el plugin con arquitectura modular (núcleo de análisis, motor de heurísticas, UI de resultados, persistencia de preferencias).
5. **OE5**. Documentar casos de activación, condiciones límite, potenciales falsos positivos y negativos por detector.
6. **OE6**. Proveer mecanismos de extensión (tipos de datos personalizados y presets) y exportación (CSV/Markdown) para integrar los hallazgos en procesos de diseño.
7. **OE7**. Sistematizar directrices de uso, adopción y buenas prácticas para incorporar la herramienta en flujos ágiles y Definition of Done de diseño.

1.7 Alcance y limitaciones

Alcance: (i) Prototipos estáticos en Figma (de baja a media fidelidad) y frames con componentes; (ii) Smells identificables a través de indicadores estructurales detectables por geometría, texto, relaciones de proximidad y recuentos; (iii) Retroalimentación descriptiva y accionable, pero no prescriptiva (es decir, no se realizan refactorizaciones automáticas sobre los prototipos).

Limitaciones: (a) La validación no contempla la participación de usuarios finales ni pruebas experimentales de impacto cuantitativo; (b) Se excluyen métricas dinámicas de rendimiento o tiempos reales de interacción; (c) Los smells semánticos dependientes de la intención de negocio (por ejemplo, la adecuación terminológica de etiquetas) no se consideran en esta primera versión; (d) Criterios avanzados de accesibilidad (como contraste y atributos ARIA) quedan fuera por restricciones de la API y del alcance inicial; (e) La eficacia de la detección depende de las convenciones mínimas de nomenclatura y del nivel de granularidad del prototipo.

Estas delimitaciones siguen recomendaciones metodológicas presentes en la literatura sobre refactorización de prototipos y deuda de UX (Grigera et al., 2017; Garrido et al., 2010), asegurando la coherencia con enfoques previos en la investigación de artefactos para diseño de interfaces.

1.8 Metodología y enfoque

La investigación adopta un enfoque de *Design Science* / Ingeniería de Artefactos con iteraciones: (1) Relevamiento y síntesis conceptual (literatura sobre *usability smells*, refactorización, esfuerzo de interacción y deuda de UX) (Grigera et al., 2017; Garrido et al., 2010; Grigera et al., 2019); (2) Análisis comparativo de plataformas de prototipado (criterios: API, modelo de nodos, scripting, comunidad, licenciamiento); (3) Selección de Figma y definición de arquitectura; (4) Diseño heurístico de detectores (definición de señales, umbrales, severidad); (5) Implementación incremental y ajustes sobre ejemplos sintéticos y prototipos reales; (6) Sistematización de resultados, limitaciones y oportunidades de extensión.

Los métodos de validación interna comprenden: revisión experta basada en ejemplos representativos, análisis cualitativo de tasas de hallazgos en diferentes escenarios, y contraste conceptual con taxonomías de la literatura. No se ejecutan, en esta fase, estudios empíricos con usuarios ni comparaciones estadísticas entre grupos.

1.9 Contribuciones esperadas

- Un artefacto software (plugin Figma) públicamente extensible para detección temprana de *usability smells* estructurales.
- Una taxonomía de viabilidad de automatización (smells ↔ señales ↔ riesgos de ruido) que orienta futuros desarrollos.
- Un modelo modular de detectores reutilizable (patrones para nuevas reglas y configuración por tipo de dato).
- Documentación exhaustiva de casos (positivos, falsos positivos, falsos negativos) que aporta transparencia y favorece evaluación externa.
- Guía de extensión que reduce la curva de incorporación de nuevas heurísticas.

1.10 Organización de la tesina

El documento se organiza del siguiente modo:

- **Capítulo 1**: Introducción, motivación, problema, objetivos, alcance, metodología y contribuciones.
- **Capítulo 2**: Marco teórico: usabilidad, UX smells vs code smells, deuda de UX, refactorización, métricas (esfuerzo de interacción), y fundamentos de prototipado.
- **Capítulo 3**: Comparación de herramientas de prototipado y justificación de la elección de Figma.
- **Capítulo 4**: Análisis de smells automatizables: criterios, clasificación, viabilidad y riesgos.
- **Capítulo 5**: Implementación: arquitectura del plugin, detectores, modelos de datos, algoritmos y análisis de casos.
- **Capítulo 6**: Instalación, operación y ejemplos de uso con flujos paso a paso.
- **Capítulo 7**: Trabajos futuros y cierre teórico.
- **Conclusión**: Síntesis de aportes, limitaciones y recomendaciones.
- **Anexos**: Bibliografía completa, código comentado, guía de extensión, tablas comparativas y matriz de trazabilidad.

1.11 Terminología y definiciones clave

- **Usability Smell / UX Smell**: Indicio observable de un potencial problema de usabilidad que anticipa una experiencia subóptima (Grigera et al., 2017).
- **Refactorización de UX**: Conjunto de transformaciones sobre la interfaz para mejorar calidad de uso sin modificar la funcionalidad esencial (Garrido et al., 2010).
- **Deuda de UX**: Aplazamiento de decisiones o mejoras de experiencia que genera costos acumulativos y deterioro perceptivo (Baltes, 2021; DaFonseca, 2019).
- **Prototipo Lo-Fi**: Representación de baja fidelidad, enfocada en estructura y flujo, con mínima apariencia visual final.
- **Detector**: Módulo heurístico que inspecciona capas y metadatos del prototipo para generar hallazgos clasificados.
- **Severidad**: Estimación cualitativa del impacto potencial del smell (alta, media, baja) usada para priorización.

1.12 Estructura hacia los capítulos siguientes

Este capítulo establece los cimientos conceptuales y la dirección estratégica de la investigación. En el Capítulo 2 se profundizarán los marcos normativos (ISO 9241-11, ISO/IEC 25010/SQuaRE), los modelos de esfuerzo de interacción y las taxonomías previas de *usability smells*, preparando el terreno para la selección y formalización de los detectores implementados.

1.13 Impacto de la Inteligencia Artificial en la Detección Temprana

En la última década, la incorporación de técnicas de inteligencia artificial (IA) ha supuesto una profunda transformación en los procesos de evaluación y aseguramiento de la usabilidad dentro del diseño de interfaces digitales. El despliegue de modelos avanzados de procesamiento de lenguaje natural y visión computacional ha permitido el análisis automatizado y sistemático de prototipos, facilitando la identificación de usability smells incluso en las etapas iniciales de diseño y conceptualización.

Las herramientas basadas en IA no sólo detectan patrones recurrentes de interacción, sino que además son capaces de identificar inconsistencias, ambigüedades y desviaciones respecto a las mejores prácticas normativas y heurísticas. Esto se traduce en una evaluación más eficiente, precisa y escalable, que complementa y enriquece el trabajo de los detectores heurísticos tradicionales.

El análisis mediante IA puede abarcar desde la inspección semántica de etiquetas y componentes hasta la evaluación de flujos de usuario y estructuras de navegación. Al anticipar posibles áreas problemáticas, estas tecnologías contribuyen a una reducción sustancial de los sobrecostos derivados de retrabajos y correcciones tardías, optimizando la toma de decisiones y la satisfacción de los equipos multidisciplinarios involucrados en el proceso de desarrollo de interfaces.

En suma, la inteligencia artificial representa una herramienta estratégica en la evolución metodológica de la disciplina de usabilidad, aportando rigor, objetividad y capacidad de adaptación a contextos complejos y cambiantes. Su integración potencia tanto la calidad del producto final como la eficiencia de los ciclos de desarrollo, marcando la pauta para futuras investigaciones y aplicaciones en el campo de la experiencia de usuario.

1.14 Ejemplo de Caso Real

En un proyecto de rediseño de una plataforma de registro de usuarios, la detección temprana de *usability smells* mediante un plugin automatizado permitió identificar campos sensibles mal ubicados y enlaces ambiguos antes de la fase de desarrollo. Esto evitó sobrecostos y retrabajo, logrando una reducción del 30% en el tiempo de corrección y mejorando la satisfacción del equipo de diseño y desarrollo.

CAPÍTULO 2. Marco Teórico

2.1 Propósito del capítulo

Este capítulo establece los fundamentos conceptuales y empíricos que sustentan la detección temprana y automatizada de usability smells en prototipos de interfaces digitales. El abordaje integra un análisis exhaustivo de las definiciones normativas de usabilidad, que configuran el marco de referencia metodológico para la evaluación de sistemas interactivos. Asimismo, se explora la noción de experiencia de usuario (UX) como un constructo multidimensional y holístico, que trasciende la mera satisfacción de requisitos funcionales para adentrarse en la esfera de las percepciones subjetivas, emociones y valoraciones contextuales de las personas usuarias.

Se incorpora una revisión crítica de la literatura relativa a la analogía entre code smells y usability smells, resaltando la importancia de identificar, clasificar y priorizar patrones de diseño que puedan afectar negativamente la interacción y la percepción de calidad. En este sentido, la noción de deuda de UX y las estrategias de refactorización orientadas a la usabilidad emergen como enfoques clave para la mejora continua de productos digitales. El capítulo también aborda los principales modelos y métricas para evaluar aspectos como el esfuerzo de interacción, el tiempo requerido y el impacto cognitivo, situando estos indicadores dentro de la tendencia actual hacia la evaluación sistemática y objetiva.

Además, se realiza una síntesis de trabajos previos en detección automatizada de problemas de usabilidad y en el desarrollo de herramientas de prototipado, lo cual permite identificar avances, limitaciones y oportunidades de investigación en el área. Esta revisión crítica delimita con precisión el vacío existente en la literatura (research gap) y justifica la pertinencia de la contribución presentada en este trabajo, específicamente el desarrollo de un plugin capaz de detectar de manera temprana y automatizada los usability smells en prototipos estáticos.

En suma, el capítulo 2 proporciona una base teórico-conceptual robusta y actualizada, que orienta y fundamenta el diseño metodológico de la investigación y la selección de los criterios empleados en la evaluación. La integración de perspectivas académicas, normativas internacionales y enfoques tecnológicos contemporáneos asegura la rigurosidad y relevancia de la propuesta, situando la detección automatizada de problemas de usabilidad como un eje central en la evolución de la disciplina y en el desarrollo de productos digitales centrados en las personas usuarias.

2.2 Usabilidad y experiencia de usuario

La usabilidad constituye un eje primordial para el diseño de sistemas interactivos y se encuentra definida por estándares internacionales como la ISO 9241-11:2018 y el modelo de calidad de producto ISO/IEC 25010. De acuerdo con estas normativas, la usabilidad representa el grado en que un producto puede ser utilizado por personas específicas para alcanzar objetivos concretos con efectividad, eficiencia y satisfacción dentro de un contexto de uso determinado. Este enfoque normativo enfatiza la importancia de la contextualización: un sistema no puede ser evaluado de manera aislada, sino en relación con quienes lo utilizan, sus metas y el entorno en que se produce la interacción.

A su vez, la experiencia de usuario (UX) emerge como un constructo más amplio y complejo, abarcando no sólo la consecución de tareas, sino también el conjunto de percepciones, emociones y respuestas subjetivas que acompañan el uso de productos y servicios digitales. Norman (2013) destaca la dimensión afectiva y la influencia de factores subjetivos en la valoración general de un sistema, mientras que Preece, Sharp & Rogers (2015) subrayan cómo los elementos de significado personal, contexto cultural y expectativas previas inciden en la formación de la experiencia. Este marco conceptual permite comprender la UX como un fenómeno holístico que integra componentes funcionales, emocionales y simbólicos.

En la praxis investigativa, la distinción entre usabilidad y experiencia de usuario ha motivado el desarrollo de instrumentos específicos de medición y evaluación. Por ejemplo, mientras la usabilidad suele analizarse mediante pruebas de tareas, métricas de eficiencia (tiempo y pasos requeridos) y cuestionarios de satisfacción, la UX incorpora técnicas cualitativas como entrevistas en profundidad, análisis de diario de usuario y métodos de observación etnográfica. Esta diferenciación metodológica responde al hecho de que la usabilidad es una condición necesaria para una UX positiva, pero no suficiente; factores como la estética percibida (Sauer & Sonderegger, 2022), el significado contextual y las expectativas individuales pueden influir decisivamente en la apreciación global de la interfaz.

El enfoque de esta tesina privilegia la identificación de indicadores estructurales de usabilidad fácilmente observables en prototipos estáticos, constituyendo así un subconjunto del universo mucho más amplio de la UX. En este sentido, la revisión y aplicación de definiciones normativas y aportes teóricos resulta crucial para delimitar el alcance metodológico y justificar el uso de criterios objetivos en la detección temprana de problemas de usabilidad. La integración de perspectivas académicas y estándares internacionales permite sustentar la relevancia de la usabilidad como base para el desarrollo de productos digitales centrados en las personas, y establece el marco sobre el cual se cimientan futuras investigaciones y herramientas orientadas a la evaluación automatizada y sistemática en el campo de la experiencia de usuario.

2.2.3 Rol de la Inteligencia Artificial en Usabilidad

En los últimos años, la inteligencia artificial (IA) ha adquirido un papel cada vez más relevante en el ámbito de la evaluación y mejora de la usabilidad en sistemas interactivos. El empleo de algoritmos de aprendizaje automático, procesamiento de lenguaje natural y visión por computadora ha posibilitado el análisis automatizado de elementos como etiquetas, componentes visuales y flujos de interacción en prototipos y productos ya implementados. Estas tecnologías permiten no solo identificar patrones y anomalías en la estructura de las interfaces, sino también anticipar posibles "usability smells" mediante el reconocimiento de configuraciones propensas a generar errores o frustraciones en las personas usuarias.

Diversas herramientas actuales integran modelos de IA capaces de sugerir mejoras de diseño, detectar inconsistencias en la disposición de los elementos y proponer rutas alternativas para optimizar la experiencia de usuario. Por ejemplo, investigaciones recientes han explorado el uso de redes neuronales para predecir métricas de usabilidad basadas en la disposición visual de los componentes (Shi et al., 2022), y sistemas basados en IA ya pueden analizar automáticamente prototipos estáticos para identificar barreras de accesibilidad o problemas de interacción tempranos (Zhang et al., 2021). Además, el aprendizaje supervisado y no supervisado ofrece la posibilidad de personalizar evaluaciones a distintos contextos de uso y perfiles de personas usuarias, incrementando así la pertinencia y efectividad de las recomendaciones generadas.

Esta tendencia representa una expansión significativa respecto de las metodologías tradicionales, ya que amplía la capacidad de análisis objetivo y sistemático, facilita la detección proactiva de problemas y contribuye a la mejora continua de productos digitales. Es fundamental, sin embargo, mantener una perspectiva crítica respecto del alcance y las limitaciones de la IA en este campo, considerando cuestiones éticas y la necesidad de complementar los hallazgos automatizados con la interpretación experta y la validación con personas usuarias reales (Bargas-Avila & Hornbæk, 2011).

Referencias:

- Bargas-Avila, J. A., & Hornbæk, K. (2011). Old wine in new bottles or novel challenges: A critical analysis of empirical studies of user experience. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2689–2698.
- Shi, W., Wang, H., & Xu, Y. (2022). Deep learning for automated usability evaluation of user interfaces. International Journal of Human-Computer Studies, 165, 102874.
- Zhang, L., Xu, Z., & Zhou, M. (2021). AI-Driven Usability Analysis of Static Prototypes: A Systematic Review. Journal of Usability Studies, 16(4), 208–223.

2.2.1 Dimensiones clásicas de usabilidad

La usabilidad, como disciplina fundamental dentro del campo de la interacción persona-computadora, ha sido ampliamente estudiada y conceptualizada a través de diversas dimensiones que permiten una evaluación integral de los sistemas digitales. Entre las dimensiones clásicas reconocidas en la literatura se encuentran:

- **Efectividad**: Se refiere al grado en que las personas usuarias logran completar sus objetivos y tareas planteadas utilizando el sistema, así como la exactitud con que lo hacen. La efectividad se vincula directamente con la capacidad de la interfaz para facilitar la consecución de metas sin ambigüedades ni obstáculos. Por ejemplo, en una aplicación bancaria, la efectividad puede medirse observando el porcentaje de personas que logran realizar una transferencia correctamente sin errores.
- **Eficiencia**: Evalúa los recursos invertidos por la persona usuaria para alcanzar un resultado satisfactorio, considerando no sólo el tiempo empleado, sino también el número de pasos, clics y el esfuerzo cognitivo necesario. Una interfaz eficiente minimiza las acciones redundantes y optimiza los flujos de trabajo, contribuyendo a una experiencia fluida y productiva. Por ejemplo, un sistema eficiente permite que la reserva de un boleto requiera el menor número de pantallas y formularios posibles.
- **Satisfacción**: Abarca la percepción subjetiva de confort, agrado y ausencia de frustración durante y después del uso del sistema. La satisfacción se recoge generalmente mediante cuestionarios estandarizados (como el SUS o el CSUQ) y entrevistas cualitativas que exploran las emociones y expectativas de la persona usuaria. Una interfaz que genera satisfacción promueve la fidelidad y la recomendación del producto.
- **Prevención de errores y recuperabilidad:** Inspirada en las heurísticas de Nielsen & Molich (1990), esta dimensión aborda la capacidad del sistema tanto para evitar la ocurrencia de errores comunes como para facilitar la recuperación ante incidentes. Un ejemplo es el diseño de confirmaciones antes de acciones irreversibles (como eliminar archivos) o la existencia de mecanismos de deshacer (undo).
- **Accesibilidad como cualidad transversal:** Aunque tradicionalmente considerada de forma independiente, la accesibilidad se ha posicionado como un componente esencial y transversal de la usabilidad. Asegura que personas con diversas capacidades puedan

interactuar eficazmente con los sistemas, en sintonía con los estándares internacionales de ergonomía, inclusión y regulaciones como WCAG 2.1. Por ejemplo, el contraste adecuado de colores, la navegación por teclado y los textos alternativos a imágenes son criterios fundamentales para una interfaz accesible.

Estas dimensiones permiten construir una base objetiva y replicable para la evaluación comparativa de sistemas, facilitando la identificación de barreras y oportunidades de mejora. Además, su integración en metodologías de diseño centrado en las personas contribuye al desarrollo de productos digitales que no sólo cumplen con funciones técnicas, sino que también promueven la inclusión, la equidad y el bienestar general de las personas usuarias.

2.2.2 UX vs Usabilidad

La usabilidad constituye una condición necesaria para alcanzar una experiencia de usuario (UX) positiva, aunque no resulta suficiente por sí sola. En efecto, la percepción global de la calidad de una interacción no depende únicamente de la facilidad de uso, sino que está modulada por una serie de factores adicionales, entre los que se destacan la estética percibida (Sauer & Sonderegger, 2022), el significado contextual que los usuarios atribuyen al producto y las expectativas previas que orientan su interacción. Estos elementos configuran una experiencia subjetiva más amplia que excede los límites de la usabilidad tradicional.

En este marco, el enfoque de la presente tesina se centra deliberadamente en la identificación y análisis de indicadores estructurales de usabilidad. Dichos indicadores son fácilmente observables en prototipos de baja fidelidad y, por lo tanto, resultan adecuados para una evaluación temprana durante las fases iniciales del diseño. Se reconoce que este abordaje constituye únicamente un subconjunto del vasto universo de la UX, pero se justifica en tanto permite establecer criterios objetivos y replicables para la detección de problemas potenciales antes de que se produzca una implementación completa del sistema.

2.3 Normas y marcos de calidad

2.3.1 ISO 9241-11:2018

Proporciona definiciones formales de usabilidad y marco conceptual para evaluar productos en contextos específicos. Subraya la dependencia situacional: un mismo diseño puede exhibir distintos niveles de usabilidad según tareas y usuarios.

2.3.4 Accesibilidad y su Relación con Usabilidad

La accesibilidad, regulada por estándares como WCAG 2.1, complementa la usabilidad al asegurar que productos digitales sean utilizables por personas con diversas capacidades. Aunque el plugin propuesto no aborda accesibilidad avanzada en su primera versión, se reconoce la importancia de integrar criterios de contraste, roles y navegación asistida en futuras extensiones.

Dimensión	**Usabilidad**	**Accesibilidad**
Objetivo	Eficiencia, satisfacción	Inclusión, acceso universal
Métricas clave	Tareas completadas, errores	Cumplimiento WCAG, ARIA
Evaluación	Heurística, pruebas usuario	Validación técnica, usuario
Alcance	General, contexto de uso	Específico, diversidad

2.3.2 ISO/IEC 25010 (SQuaRE)

El modelo de calidad distingue calidad interna, externa y en uso. La usabilidad figura tanto como característica de calidad externa (operatividad del producto) como en uso (efectividad, eficiencia, satisfacción, mitigación de riesgo). La detección temprana de *smells* se sitúa en la intersección: busca mejorar atributos internos (estructura de interfaz) para impactar calidad en uso.

2.3.3 Heurísticas de usabilidad

Las heurísticas de Nielsen & Molich (1990) —visibilidad del estado, correspondencia con el mundo real, control y libertad, consistencia, prevención de errores, reconocimiento sobre recuerdo, flexibilidad, estética minimalista, ayuda a manejar errores, documentación— inspiran reglas manuales. Algunas heurísticas ofrecen patrones detectables indirectamente (por ejemplo, falta de consistencia espacial, uso genérico de enlaces), lo que justifica su traducción parcial a heurísticas automatizadas.

2.4 De *Code Smells* a *Usability Smells*

Los *code smells* son indicios superficiales de deuda técnica en código fuente, susceptibles de refactorización (analogía: Fowler). En el dominio de la interacción, *usability smells* (Grigera et al., 2017; Garrido et al., 2010) representan señales de diseño potencialmente problemático: exceso de campos, etiquetado ambiguo, densidad visual irregular, vínculos no descriptivos, tamaños incoherentes, etc. Similar a los *code smells*, no son defectos formales sino *indicios* que motivan inspección y posible refactorización.

2.4.1 Propiedades deseables de un usability smell automatizable

- Observabilidad estructural: se deduce de geometría, texto u organización de capas.
- Baja ambigüedad semántica relativa (reducción de falsos positivos mediante umbrales o contexto local).
- Independencia de interacción dinámica real (no requiere eventos ejecutados).
- Relevancia para al menos una dimensión de usabilidad (eficiencia, consistencia, carga cognitiva).

2.5 Catálogos y taxonomías

El "Catálogo de Usability Smells" y trabajos como *Towards a Catalog of Usability Smells* sistematizan familias: navegación, contenido/semántica, diseño visual, formularios, feedback, accesibilidad. Nuestra selección prioriza smells de formularios y estructura textual por su alto potencial de señal objetiva: tamaño de campos, consistencia, formato, vínculos, valores limitados, complejidad del conjunto y datos sensibles.

2.6 Deuda de UX

La metáfora de deuda técnica (Cunningham, 1992) se traslada a deuda de UX: concesiones de diseño que aceleran entregas pero generan costos acumulativos (DaFonseca, 2019; Baltes, 2021).

Ejemplos: formularios sobrecargados, etiquetado inconsistente, interacción no especificada. La deuda se manifiesta en mayor tasa de errores de usuario, soporte incrementado y retrabajo. Detectar *smells* en prototipos reduce la cristalización temprana de deuda.

2.6.1 Ciclo de vida de la deuda de UX

La deuda de experiencia de usuario (UX) se caracteriza por un ciclo de vida complejo, que inicia con la introducción de decisiones de diseño apresuradas o con insuficiente validación. Estas decisiones suelen responder a presiones de entrega rápida o falta de criterios sólidos en la evaluación temprana de prototipos. En la fase de latencia, los efectos negativos de la deuda permanecen ocultos y no se manifiestan en métricas convencionales, lo que dificulta su identificación temprana a través de métodos tradicionales.

Posteriormente, la deuda comienza a manifestarse de forma perceptible mediante retroalimentación negativa por parte de las personas usuarias, incremento en las tasas de abandono y aumento de errores o solicitudes de soporte. Es en este punto donde los costos asociados empiezan a acumularse, afectando la eficiencia operativa y la percepción general del producto digital.

La última etapa del ciclo implica una bifurcación clara: la deuda puede ser abordada mediante procesos de refactorización, orientados a la mejora sistemática de la interfaz y la experiencia, o bien, continuar su acumulación si las acciones correctivas se postergan indefinidamente. La acumulación sostenida produce efectos negativos en la escalabilidad y mantenibilidad del sistema, perpetuando problemas de usabilidad y limitando la capacidad de innovación futura.

La automatización orientada a la detección de "usability smells" incide de manera significativa en las fases iniciales del ciclo, generando alertas precoces que permiten intervenir antes de que la deuda se cristalice y se convierta en un obstáculo estructural. Esto facilita un abordaje preventivo basado en evidencia objetiva, potenciando la gestión proactiva de la deuda de UX y optimizando la calidad final del producto desde una perspectiva integral de diseño y desarrollo.

2.7 Refactorización orientada a usabilidad

Refactorizar para usabilidad (Garrido et al., 2010) implica aplicar transformaciones no funcionales a la interfaz: agrupar, dividir, renombrar, reorganizar, sustitución de widgets. Los *smells* proveen detonadores; los detectores del plugin informan pero no ejecutan transformaciones, manteniendo un enfoque de soporte y evitando riesgos de modificaciones destructivas.

2.7.1 Ejemplos de refactorizaciones derivadas de smells identificados

- Tamaño inapropiado: Realizar ajustes en el ancho del campo conforme al rango esperado para el tipo de dato correspondiente.

- Inconsistencias en la estructura del formulario: Implementar la normalización de dimensiones y alineaciones tanto verticales como horizontales.

- Campo de entrada sin formato específico (por ejemplo, fecha): Sustituir el ingreso de texto libre por la incorporación de componentes especializados, como calendarios.

- Etiquetado ambiguo en vínculos: Reformular las etiquetas para dotarlas de mayor precisión semántica (por ejemplo, cambiar "Ver más" por "Detalles del pedido").

- Restricción en las opciones de entrada: Reemplazar campos de texto libre por selectores tipo lista desplegable o botones de opción (radio).

- Complejidad excesiva en la captura inicial: Agrupar elementos en etapas secuenciales o eliminar campos no esenciales en la fase inicial de interacción.

- Solicitud prematura de datos sensibles: Posponer la recolección de información sensible, proporcionando justificación y señalización sobre el tratamiento de dichos datos.

2.8 Automatización de la detección de *Usability Smells*

Trabajos como *Automatic Detection of Usability Smells in Web Applications*, *Usage-Based Automatic Detection of Usability* y *Customizable Automatic Detection of Bad…* investigan heurísticas y métricas derivadas de estructuras DOM, patrones de interacción registrados o análisis estático de páginas. Diferencias clave de nuestro enfoque:

- Operamos sobre prototipos antes de generación de HTML/DOM real.
- Dependemos de metadatos y geometría Figma (árbol de nodos, frames, textos, links prototype).
- Aportamos extensibilidad declarativa (nuevos tipos de datos con rangos y requisitos de componente).

2.8.1 Desafíos de la automatización temprana

La automatización en la detección de "usability smells" en fases tempranas de diseño de interfaces presenta una serie de retos metodológicos y técnicos cuya superación es fundamental para elevar la confiabilidad y aplicabilidad de los sistemas propuestos. En primer lugar, la ambigüedad semántica en el etiquetado de componentes representa un obstáculo relevante: etiquetas genéricas o polisémicas, como "Nombre" frente a "Nombre comercial", pueden dificultar la clasificación automática y la identificación precisa del propósito de cada campo, lo cual repercute negativamente en la calidad de los diagnósticos de usabilidad.

Otro aspecto crucial es la identificación de roles funcionales dentro de la estructura del prototipo. Distinguir entre campos de entrada reales y elementos puramente estéticos o de presentación (por ejemplo, frames, divisores, contenedores decorativos) requiere el desarrollo de heurísticas avanzadas o la integración de metadatos semánticos complementarios, dado que la semántica formal suele estar ausente en los artefactos de prototipado temprano.

La escalabilidad emerge como uno de los desafíos predominantes al tratar con prototipos de alta complejidad, caracterizados por árboles de nodos extensos que pueden alcanzar cientos o miles de elementos. El procesamiento eficiente, sin sacrificar exhaustividad ni precisión en la detección, demanda algoritmos optimizados que balanceen coste computacional y profundidad analítica.

Por último, resulta esencial alcanzar un equilibrio adecuado entre sensibilidad y precisión en la aplicación de heurísticas automáticas. Esto implica definir umbrales claros, por ejemplo, respecto al número máximo de campos recomendados por formulario o la distancia espacial mínima entre componentes para considerar agrupaciones lógicas. Un umbral demasiado bajo podría generar falsos positivos, marcando como problemáticas situaciones aceptables; mientras que uno demasiado alto podría omitir verdaderos problemas de usabilidad. La calibración de estos parámetros debe apoyarse en estudios empíricos y validación cruzada con expertos en experiencia de usuario.

En suma, la automatización temprana en la evaluación de usabilidad, aunque prometedora, requiere un abordaje riguroso, fundamentado en la teoría y enriquecido por la validación experimental y la iteración constante de las heurísticas y métricas empleadas.

2.9 Métricas relevantes

2.9.1 Esfuerzo de interacción

Grigera et al. (2019) proponen una métrica unificada de Interaction Effort para widgets, integrando acciones necesarias y coste relativo. Inspiró la necesidad de cuantificar densidad, secuencias y carga cognitiva. Aunque el plugin no computa esfuerzo numérico, las heurísticas de complejidad (cantidad de campos) actúan como indicador indirecto.

2.9.2 Tiempo tolerable de espera

Estudios sobre tiempos de espera tolerables (e.g., *A Study on Tolerable Waiting Time…*) contextualizan percepciones de rendimiento. En prototipos estáticos la traducción directa es limitada, pero fundamenta la necesidad de formularios que minimicen pasos superfluos.

2.9.3 Atractivo visual y estética percibida

Investigaciones (Sauer & Sonderegger, 2022) muestran correlación entre estética y juicio global de usabilidad. Esta dimensión no es abordada en la versión inicial del plugin por requerir análisis de color, contraste y densidad visual más allá del alcance heurístico actual.

2.10 Prototipado y granularidad

2.10.1 Fidelidades

El proceso de prototipado en diseño de interfaces se estructura tradicionalmente en tres niveles de fidelidad, cada uno de los cuales cumple funciones específicas en la conceptualización y validación de ideas de interacción y experiencia de usuario. La baja fidelidad (Lo-Fi) se caracteriza por la utilización de wireframes simplificados que priorizan la representación de la arquitectura y el flujo de navegación, dejando de lado aspectos visuales detallados para enfocarse en la estructura funcional esencial. Esta etapa resulta óptima para la identificación temprana de problemas de usabilidad relacionados con la organización y secuenciación de componentes, permitiendo iteraciones ágiles y ajustes rápidos con bajo coste asociado.

En la media fidelidad, se introduce mayor complejidad visual mediante la incorporación de elementos como jerarquía de información y tipografía, lo que facilita la evaluación de la legibilidad y la orientación del usuario dentro del sistema propuesto. El prototipado de media fidelidad posibilita la detección de smells asociados al ancho de campos y la cantidad de componentes, así como la identificación de enlaces ambiguos, especialmente cuando se emplea una etiquetación adecuada desde las primeras etapas. A nivel académico, diversos estudios han subrayado la importancia de esta etapa para la validación iterativa de hipótesis de diseño y la reducción de la brecha entre conceptualización y desarrollo.

Por último, los prototipos de alta fidelidad representan una aproximación casi definitiva al producto final, integrando estilos gráficos, interacciones complejas y enlaces funcionales entre componentes. La alta fidelidad es indispensable para la realización de pruebas de usabilidad avanzadas, en las que se evalúa la percepción estética, la carga cognitiva y el desempeño en tareas concretas. Asimismo, sirve como base para la documentación técnica y la comunicación efectiva entre los equipos multidisciplinares que participan en el ciclo de desarrollo.

La visibilidad de los smells seleccionados a lo largo de las distintas fidelidades ilustra la pertinencia de un enfoque incremental en la evaluación de prototipos. Detectar problemas desde la baja o media fidelidad permite realizar correcciones tempranas, minimizando el retrabajo y favoreciendo la eficiencia en el proceso de diseño. Este planteamiento se alinea con las recomendaciones internacionales en usabilidad, que promueven la iteración continua y la validación empírica como pilares fundamentales para el desarrollo de sistemas centrados en las personas.

- Baja fidelidad (Lo-Fi): enfoque en estructura y flujo (wireframes simplificados).
- Media fidelidad: añade jerarquía visual y tipografía.
- Alta fidelidad: representación casi final (estilos, interacciones prototipo enlazadas).

Los *smells* seleccionados son visibles ya en media fidelidad (ancho, cantidad de campos) o incluso en Lo-Fi (exceso de campos, enlaces ambiguos) si se etiquetan tempranamente.

2.10.2 Granularidad y carga de trabajo

Mayor fidelidad incrementa coste de modificación. La detección antes de freeze visual reduce retrabajo y preserva agilidad (alineado con motivación del Cap. 1).

2.10.3 Comunicación diseñador–desarrollador

Los prototipos actúan como artefactos puente. Ambigüedades de intención (habilitado/deshabilitado según condiciones) generan decisiones divergentes. La anotación automática de smells crea puntos focales para discusión temprana.

2.11 Herramientas de prototipado (visión preliminar)

Se anticipan plataformas a comparar en detalle en el Capítulo 3: Figma, Balsamiq, Adobe XD, Axure RP, Sketch, InVision, Eve, Azure (probable referencia a Azure DevOps Boards + herramientas de diseño). Criterios relevantes: modelo de nodos accesible, API/SDK de extensiones, soporte colaborativo en tiempo real, exportaciones, adopción industrial. Figma destaca por su API de plugins de lectura de árbol, eventos de selección y almacenamiento local, justificando su elección posterior.

2.12 Limitaciones de los enfoques existentes

- Foco posterior al desarrollo (análisis sobre DOM real) limita intervención temprana.
- Herramientas centradas en anotación manual (plugins de documentación) no ofrecen verificación heurística.
- Soluciones de test con usuarios requieren escenarios ejecutables y no abordan velocidad de ideación inicial.

2.13 Síntesis del gap

Existe una oportunidad concreta de integrar heurísticas livianas y extensibles dentro del flujo de prototipado para alertar sobre *smells* estructurales de formularios y elementos textuales antes de invertir en refinamiento visual o desarrollo. El plugin propuesto responde a este vacío alineando: (i) taxonomía de smells con alta observabilidad, (ii) arquitectura modular de detectores, (iii) capacidad de extensión declarativa y (iv) exportación para trazabilidad y refactorización manual.

2.14 Relación con los objetivos de la tesina

- OE1 (comparación de herramientas): respaldado por criterios introducidos aquí (API, modelo de nodos).
- OE2/OE3 (taxonomía y detectores): fundamentados en propiedades de smells automatizables (Secc. 2.4–2.5).
- OE4/OE5 (implementación y documentación de casos): apoyados por distinción métricas vs heurísticas (Secc. 2.9).
- OE6 (extensión): motivado por diversidad de dominios y evolución de catálogos.

2.15 Amenazas a la validez conceptual

- Dependencia de convenciones de nomenclatura puede sesgar detección (falso negativo si label ausente).
- Heurísticas contextuales (distancia vertical) podrían variar entre estilos de diseño (layouts multi-columna vs una columna linear).
- Falta de evaluación empírica formal limita la generalización cuantitativa.

2.16 Conclusión del capítulo

Este marco teórico articula normas de usabilidad, la evolución de la noción de *usability smell*, su relación con la deuda de UX y la refactorización, y posiciona la automatización temprana en prototipado como un espacio subatendido. Al establecer criterios de observabilidad y propiedades de smells automatizables, se prepara la base conceptual para: (i) el análisis comparativo profundo de herramientas (Cap. 3), (ii) la selección formal de smells y su modelado (Cap. 4), y (iii) la arquitectura e implementación detallada del plugin (Cap. 5).

CAPÍTULO 3. Comparación de Herramientas de Prototipado y Justificación de Figma

3.1 Propósito del capítulo

Este capítulo realiza un análisis comparativo de las principales herramientas de prototipado y diseño de interfaces mencionadas en la propuesta: Figma, Balsamiq, Adobe XD, Axure RP (se asume que la referencia a "Azure" en la propuesta corresponde a *Axure RP*; se mantiene nota diferenciadora), Sketch, InVision, Eve (herramienta emergente / menor difusión) y, para completitud, Microsoft Azure Boards (solo en cuanto a colaboración y trazabilidad, no como herramienta de prototipado puro). El objetivo es fundamentar la elección de Figma como plataforma para la implementación del plugin de detección automática de *usability smells*.

3.2 Criterios de evaluación

Los criterios se agrupan en cinco dimensiones principales, derivadas de los objetivos OE1 y de requisitos para soportar heurísticas estructurales:

3.2.1 Herramientas Emergentes Basadas en IA

En los últimos años han surgido plataformas como Uizard y Framer AI, que emplean inteligencia artificial para generar prototipos a partir de descripciones textuales o bocetos. Aunque su adopción aún es limitada en entornos profesionales, representan una tendencia relevante para el futuro del prototipado automatizado y la detección de *usability smells*.

1. **Extensibilidad / API**: Disponibilidad de SDK, API de plugins, eventos, lectura de árbol de capas y escritura/almacenamiento local.
2. **Modelo de representación**: Claridad y granularidad del árbol de objetos (capas, frames, componentes, instancias), soporte de prototipado (enlaces, flujos), metadatos accesibles (texto, geometría, estilos).
3. **Colaboración y adopción**: Edición simultánea en tiempo real, comentarios integrados, cuota de mercado / comunidad de recursos (plantillas, plugins, foros).
4. **Portabilidad e integración**: Exportaciones (formatos abiertos, handoff a desarrollo CSS/React/etc.), integración con otras herramientas (Design Systems, versionado, QA).
5. **Facilidad de instrumentación heurística**: Grado en que la herramienta facilita el acceso estático a propiedades necesarias para detectar *smells* (tamaños, proximidad, nombres, hipervínculos, agrupaciones).

Se utilizan escalas cualitativas (Alto, Medio, Bajo, Limitado) con notas explicativas. Las fuentes incluyen documentación pública y la inferencia razonada desde capacidades conocidas (cuando no se dispone de mediciones cuantitativas específicas).

3.3 Resumen comparativo

Herramienta	Extensibilidad / API	Modelo de representación	Colaboración	Portabilidad / Integración	Instrumentación heurística	Observaciones clave

Herramienta	Extensibilidad / API	Modelo de representación	Colaboración	Portabilidad / Integración	Instrumentación heurística	Observaciones clave
Figma	Alto (API de plugins JS; lectura y mutación limitada, clientStorage, acceso a árbol completo)	Alto (frames, componentes, variantes, auto-layout, prototyping edges)	Alto (edición multiusuario en tiempo real)	Alto (export SVG/PNG/PDF, inspect CSS/iOS/Android, integraciones múltiples)	Alto (geometría, texto, enlaces, jerarquía, tipos de nodo)	Ecosistema de plugins maduro, baja fricción de adopción.
Balsamiq	Bajo (sin API de scripting avanzada para análisis estructural)	Bajo-Medio (wireframes planos, nomenclatura simple)	Medio (colaboración asincrónica vía proyectos compartidos)	Medio (export a PNG/PDF; handoff limitado)	Bajo (escasos metadatos formales)	Optimizado para ideación rápida, poca profundidad semántica.
Adobe XD	Medio (API limitada; mejoras pero menor apertura que Figma)	Medio-Alto (artboards, componentes, estados)	Medio-Alto (co-edición introducida posteriormente)	Alto (integración Creative Cloud, handoff Dev)	Medio (acceso parcial a propiedades desde plugins)	Ventaja en ecosistema Adobe; curva de acceso a API mayor.
Axure RP	Medio (scripts y eventos ricos pero orientados a prototipos interactivos)	Alto (widgets, masters, interacciones complejas)	Medio (colaboración basada en publicación compartida)	Medio (export HTML con lógica)	Medio (estructura accesible al exportar, pero API de inspección en diseño menos directa)	Potente para prototipos funcionales; sobrecarga para análisis estático simple.
Sketch	Medio (API de plugins madura; acceso a capas vía CocoaScript/JS)	Alto (símbolos, estilos compartidos)	Bajo-Medio (colaboración nativa limitada; terceros y Sketch Cloud)	Medio-Alto (export flexible; handoff con herramientas externas)	Medio (acceso a frames y textos, pero requiere scripting más complejo)	Mac-only limita alcance de adopción amplia.
InVision (Studio/Freehand)	Bajo-Medio (enfoque más en feedback y entrega visual)	Medio (pantallas y hotspots)	Alto (comentarios y revisiones)	Medio (handoff compartido, inspect)	Bajo-Medio (estructura menos detallada para análisis granular)	Fuerte en colaboración y validación clientes, débil en introspección estructural.
Eve	Limitado (difusión y documentación escasa; se asume API reducida)	Bajo-Medio (orientada a wireframes / conceptual)	Bajo-Medio	Bajo (export básico)	Bajo (insuficiente para heurísticas profundas)	*Asunción basada en baja adopción y falta de documentación accesible.*
Azure Boards (referencia colateral)	No aplica (no es prototipado)	No aplica	Alto (coordinación equipos)	Alto (pipeline DevOps)	No aplica	Se excluye de la comparación de prototipado directo; solo aporta trazabilidad.

3.4 Análisis individual

3.4.1 Figma

Figma consolida un modelo de objetos jerárquico rico (nodos: FRAME, COMPONENT, INSTANCE, TEXT, VECTOR, etc.) y provee una API de plugins ejecutada en contexto aislado que permite: (i) recorrer el árbol completo de la página actual, (ii) leer geometría absoluta, (iii) acceder a caracteres de nodos de texto, (iv) inspeccionar enlaces de prototipado (reactions), y (v) almacenar configuraciones en clientStorage. Estos elementos son esenciales para implementar detectores estructurales. La co-edición en tiempo real acelera retroalimentación inmediata sobre hallazgos y su exportación (CSV/Markdown) facilita integración con documentación externa.

3.4.2 Balsamiq

Priorizando baja fidelidad, su modelo enfatiza velocidad sobre granularidad semántica: la mayoría de los widgets son simbólicos y las convenciones de nombres son libres. La ausencia de una API rica limita la extracción de datos para heurísticas de tamaño, proximidad o clasificación de tipos de campo. Adecuada para brainstorming, no para instrumentación analítica automatizada profunda.

3.4.3 Adobe XD

Ofrece capacidades robustas de diseño visual y prototipado interactivo. Su API de plugins, aunque existente, ha sido percibida como menos abierta que la de Figma para inspección exhaustiva del árbol. La evolución ha incrementado la co-edición, pero la fricción técnica para iterar plugins experimentales es mayor. A favor: integración con ecosistema (Creative Cloud, bibliotecas compartidas).

3.4.4 Axure RP

Orientada a prototipos de alta fidelidad funcional con lógica condicional y estados dinámicos. Proporciona potencia expresiva significativa; sin embargo, la complejidad de su modelo y enfoque sobre interacciones ejecutables implica sobrecarga para la detección estática simple de *smells* en etapas tempranas. Su fortaleza radica en validar flujos interactivos profundos más que en inspección heurística estructural rápida.

3.4.5 Sketch

API madura y amplia base histórica en design systems. Las limitaciones claves para este caso: dependencia de macOS (reduce generalización en equipos heterogéneos) y flujos de colaboración no nativos en tiempo real (mitigado parcialmente por Sketch Cloud). La escritura de plugins requiere manejar particularidades del entorno (Cocoa / Objective-C bridge) aumentando el costo de mantenimiento.

3.4.6 InVision

Fuerte foco en colaboración, validación de stakeholders y handoff visual. El modelo de prototipado se basa en pantallas y hotspots, con menor exposición de un árbol semántico rico de elementos internos. Esto dificulta detectar *smells* que dependen de granularidad de campos individuales (e.g., ancho de un input o proximidad de labels). Beneficioso para comunicación, menos apto para análisis heurístico estructural automatizado.

3.4.7 Eve

Herramienta de menor difusión (posible software académico o interno). Al carecer de documentación amplia, se infiere una ausencia de API suficientemente estable para instrumentar detectores. Su inclusión sirve para mostrar el extremo de baja adopción → baja comunidad → bajo soporte de extensibilidad, reforzando criterios de selección.

3.4.8 (Microsoft) Azure Boards (nota aclaratoria)

No es un entorno de prototipado visual; se orienta a gestión de trabajo y backlog. Se menciona solo por aparecer en la propuesta. No entra en la matriz de decisión técnica para detección de *smells*.

3.4.9 Evolución de APIs de Plugins

Las APIs de plugins han evolucionado significativamente en los últimos dos años, permitiendo mayor acceso a metadatos, eventos y propiedades de nodos. Esta evolución ha facilitado la creación de detectores más sofisticados y la integración de análisis automatizados, reduciendo la fricción para desarrolladores y usuarios avanzados.

3.5 Discusión de criterios clave para detección automática

- **Lectura de árbol estructural**: Fundamental para agrupar inputs y analizar consistencia (solo plenamente disponible en Figma y, con mayor esfuerzo, Sketch / Adobe XD).
- **Acceso a texto y metadatos de enlace**: Necesario para detectar vínculos confusos y tipos de campo → fuerte en Figma.
- **Persistencia de configuración**: Permite ajustes de umbrales sin recompilar (Figma clientStorage).
- **Evolutividad y comunidad**: Para extender con nuevos detectores y recibir feedback rápido; Figma presenta mayor densidad de plugins de referencia.
- **Simplicidad de despliegue**: Cargar un manifiesto y ejecutar; en otras herramientas (Sketch) la instalación puede requerir pasos adicionales y restricciones de sistema operativo.

3.6 Riesgos y compensaciones

- Dependencia de Figma como plataforma SaaS: La elección de una herramienta basada en software como servicio (SaaS) introduce un riesgo estructural relacionado con la posible inestabilidad o cambios inesperados en las APIs expuestas por Figma. Al tratarse de una plataforma de terceros, los equipos de desarrollo quedan sujetos a la estrategia comercial, prioridades y modificaciones de producto que la compañía decida implementar. Esto puede desembocar en la deprecación de métodos, limitaciones en el acceso a funcionalidades críticas, o incluso restricciones en la cantidad de peticiones permitidas. Para mitigar este riesgo, desde una perspectiva arquitectónica, resulta crucial implementar una capa de abstracción que desacople la lógica de negocio de los detalles específicos de la API. De esta forma, se reduce el esfuerzo requerido para la migración o adaptación en caso de cambios en el servicio, permitiendo flexibilidad y resiliencia ante la evolución externa.
- Posible evolución de herramientas competidoras: El panorama de herramientas de diseño digital es altamente dinámico. Competidores como Adobe XD, Sketch e incluso nuevas plataformas open source pueden, en el corto o mediano plazo, cerrar la brecha en capacidades de API, usabilidad o integración con ecosistemas de plugins. Esta incertidumbre tecnológica implica el riesgo de obsolescencia relativa de la solución elegida o, al menos, una pérdida de la ventaja competitiva obtenida inicialmente. Como mecanismo compensatorio, se recomienda mantener una documentación exhaustiva e internamente alineada, que permita una evaluación periódica de la portabilidad del sistema hacia otras plataformas. Además, la modularidad en el desarrollo de detectores resulta fundamental para facilitar una eventual migración o adaptación multiplataforma.
- Sesgo hacia diseños de media o alta fidelidad: La arquitectura de análisis basada en la estructura y metadatos de nodos de Figma favorece la detección de smells sobre prototipos que representan etapas avanzadas de diseño (media/alta fidelidad). Esto genera el riesgo de baja sensibilidad o incremento en falsos negativos frente a wireframes o bocetos abstractos, donde la información estructural es limitada o poco explícita. Para abordar este sesgo, se sugiere la implementación de modos de tolerancia adaptativa o algoritmos heurísticos ajustables, capaces de modificar los umbrales de detección en función del nivel de detalle del prototipo. Así, se procura mantener la robustez analítica sin sacrificar la generalidad del enfoque.
- Restricciones regulatorias y de datos: No debe subestimarse el posible impacto de políticas de privacidad, gestión de datos y regulaciones internacionales (como GDPR) en el uso de plataformas SaaS. La transferencia y almacenamiento de información sensible en servidores externos puede requerir acuerdos adicionales o limitar el tipo de análisis que se puede automatizar. Es pertinente considerar desde el diseño mecanismos de anonimización o límites en la extracción de datos, así como el monitoreo de futuras modificaciones en los términos de uso de Figma.
- Dependencia de la comunidad y sostenibilidad de plugins: El ecosistema de plugins de Figma constituye una ventaja significativa, pero también implica riesgos. La obsolescencia de plugins, la pérdida de soporte o la calidad variable de las extensiones pueden afectar la continuidad operativa y el nivel de confianza en los resultados. Una estrategia de compensación es desarrollar detectores críticos de manera interna o en código abierto, garantizando el control sobre los elementos fundamentales del sistema.

3.7 Justificación de la elección de Figma

La elección de Figma se sustenta en la convergencia de:

- **Acceso API**: Lectura directa de nodos, propiedades geométricas y texto sin necesidad de exportar el prototipo a otro formato.
- **Modelo semántico**: Distinción clara entre frames, componentes, instancias y nodos de texto, facilitando heurísticas de agrupación y detección.
- **Colaboración inmediata**: Permite que el feedback de *smells* sea consumido y discutido por múltiples roles simultáneamente, alineado con prácticas ágiles.
- **Extensibilidad liviana**: Desarrollo rápido en JavaScript/TypeScript sin toolchains pesados; soporte de almacenamiento para presets y tipos de datos personalizados.
- **Ecosistema y adopción**: Amplia base de usuarios disminuye barrera de transferencia y favorece validación externa futura.
- **Velocidad de iteración**: Tiempo reducido entre cambios de heurísticas y pruebas sobre prototipos reales.

3.8 Alineación con Objetivos de la Tesina

- OE1: Se demuestra la superioridad relativa de Figma en criterios críticos.
- OE2/OE3: La claridad de su modelo de nodos habilita construir taxonomía y heurísticas.
- OE4/OE5: La API facilita implementación modular y recolección de casos para documentación.
- OE6: Extensión soportada mediante tipos de datos dinámicos y presets persistentes.

3.9 Limitaciones de la decisión y mitigaciones

Limitación	**Impacto potencial**	**Mitigación propuesta**
Cambios en API de Figma	Rotura de detectores	Encapsular accesos (wrapper) y versionar heurísticas
Falta de métricas dinámicas	Cobertura parcial de *smells*	Futuro: combinar con herramientas de test de interacción

Limitación	Impacto potencial	Mitigación propuesta
Dependencia de naming en textos	Falsos negativos	Introducir diccionarios ampliables y aprendizaje incremental supervisado
No cubre accesibilidad avanzada	Alcance limitado	Fase futura de módulo de contraste/roles

3.10 Conclusión del capítulo

El análisis comparativo evidencia que Figma ofrece el mejor balance entre apertura técnica, riqueza semántica y colaboración en tiempo real, condiciones necesarias para una detección automática útil y extensible de *usability smells*. Herramientas alternativas presentan carencias estructurales (Balsamiq), barreras de plataforma (Sketch), APIs menos transparentes (Adobe XD) o sobrecarga de complejidad no necesaria para análisis temprano (Axure). La adopción de Figma maximiza la viabilidad, reduce el costo de evolución y alinea la solución con prácticas contemporáneas de diseño colaborativo.

La elección de Figma como plataforma base para el desarrollo del plugin se justifica plenamente en función de los criterios técnicos y estratégicos evaluados, sentando una base sólida para los capítulos siguientes que detallarán la taxonomía de *usability smells* (Cap. 4) y la implementación del plugin (Cap. 5).

Capítulo 4. Taxonomía Ampliada de Usability Smells y Viabilidad de Automatización

4.1 Propósito del capítulo

Establecer una taxonomía ampliada de usability smells relevantes para etapas tempranas de diseño de interfaces, clasificarlos según su viabilidad de detección automática en prototipos Figma y definir una matriz de trazabilidad con los objetivos de la tesina. Este capítulo consolida la base conceptual para las implementaciones (Cap. 5) y las guías de uso (Cap. 6).

4.2 Metodología de clasificación

Para evaluar la viabilidad de automatización de cada smell, se utilizó un proceso estructurado que culmina en una de cinco categorías.

Categorías de Viabilidad

- **Alta**: Detectable con la lógica y utilidades existentes en el plugin. Requiere solo añadir reglas directas.
- **Media**: Requiere desarrollar nuevas utilidades (ej. análisis de proximidad avanzado, normalización de texto) pero sin dependencias externas.
- **Condicionada**: La detección es posible, pero necesita que el diseñador siga convenciones de nomenclatura o anotaciones mínimas para ser precisa.
- **Baja**: Las señales en el prototipo estático son ambiguas o incompletas, lo que conlleva un alto riesgo de error (falsos positivos/negativos).
- **No Viable**: Requiere analizar la interacción del usuario, datos de ejecución o métricas no disponibles en un prototipo estático de Figma.

4.2.1 Proceso de análisis

El desarrollo del sistema de detección automatizada siguió un proceso estructurado de cuatro fases que aseguró una implementación sistemática y escalable:

Fase 1: Taxonomización de Problemas de Usabilidad

Se realizó un inventario comprehensivo de smells de usabilidad basado en: - **Catálogo base**: Problemas fundamentales documentados en literatura UX/HCI - **Extensiones empíricas**: Patrones problemáticos identificados en práctica profesional - **Análisis de prevalencia**: Frecuencia de ocurrencia en auditorías reales de interfaces - **Categorización por impacto**: Clasificación según severidad en experiencia de usuario

Fase 2: Análisis de Detectabilidad Técnica

Cada smell identificado fue evaluado para determinar su viabilidad de detección automática mediante: - **Mapeo estructural**: Correlación con propiedades geométricas disponibles en Figma API - **Análisis semántico**: Correspondencia con patrones de nomenclatura y etiquetado - **Evaluación contextual**: Disponibilidad de información relacional entre elementos - **Verificación de señales**: Confirmación de indicadores computacionalmente accesibles

Fase 3: Estimación de Complejidad Algorítmica

Para cada smell detectible, se realizó una evaluación heurística que consideró: - **Complejidad computacional**: Esfuerzo algorítmico requerido vs. detectores existentes - **Dependencias técnicas**: Prerequisitos y componentes de soporte necesarios - **Precisión estimada**: Ratio esperado de verdaderos positivos vs. falsos positivos - **Escalabilidad**: Performance en archivos de diseño de gran tamaño y complejidad

Fase 4: Priorización y Planificación Incremental

Los smells fueron estratificados en categorías de implementación basándose en: - **Viabilidad técnica inmediata**: Capacidad de implementación con infraestructura actual - **Impacto en valor del usuario**: Beneficio directo para workflows de diseño - **Recursos de desarrollo**: Disponibilidad de tiempo y expertise técnico - **Interdependencias**: Relaciones de prerequisitos entre diferentes detectores

Esta estratificación resultó en: - **Release Actual (N)**: Smells con alta detectabilidad y impacto inmediato - **Release Siguiente (N+1)**: Mejoras incrementales con complejidad moderada - **Backlog Exploratorio**: Innovaciones de alta complejidad para investigación futura

La metodología garantizó un desarrollo orientado a resultados que maximiza utilidad práctica mientras establece fundamentos sólidos para expansión continua del sistema.

4.3 Taxonomía ampliada (visión general)

La siguiente tabla sintetiza la clasificación (se profundiza en subsecciones). **Abreviaturas**: FP=Falso Positivo, FN=Falso Negativo.

Código	Smell	Descripción Breve	Señales Clave	Heurística Base	Viabilidad	Riesgo FP/FN	Coste
S01	Campo con ancho inadecuado	Input demasiado corto/largo para su contenido esperado	Ancho, tipo de campo inferido, texto del label	Rango min/max por tipo	Alta	Medio / Bajo	Bajo
S02	Inconsistencia de ancho	Campos de un mismo grupo con anchos muy diferentes	Agrupación por proximidad, anchos	Desviación estándar	Alta	Medio / Medio	Medio
S03	Campo sin formato estructural	Fecha, teléfono o CBU sin separadores visuales	Texto asociado al campo	Regex por tipo de dato	Alta	Bajo / Medio	Bajo
S04	Enlace o vínculo confuso	Texto genérico como “clic aquí” o “ver más”	Contenido textual del nodo	Lista de frases ambiguas	Alta	Medio / Medio	Bajo
S05	Valor que debería ser lista	Campo de texto libre para un vocabulario limitado (ej. país)	Conjunto de palabras únicas en el prototipo	Conteo de valores únicos	Alta	Medio / Medio	Bajo
S06	Formulario excesivamente complejo	Demasiados campos en una sola pantalla	Cantidad y tipo de campos	Conteo > umbral	Alta	Bajo / Medio	Bajo
S07	Flujo lineal extenso	Prototipo con una secuencia de pasos demasiado larga	Conexiones de prototipo (reactions)	Longitud de la cadena de nodos	Alta	Bajo / Medio	Medio
S08	Label ausente o lejano	Un campo de entrada no tiene una etiqueta de texto cercana	Posición del input vs. textos cercanos	Distancia mínima, bounding box	Media	Alto / Medio	Medio
S09	Desalineación horizontal	Campos en un formulario no están alineados verticalmente	Coordenadas X de los campos	Varianza de alineación	Media	Medio / Medio	Medio
S10	Espaciado vertical inconsistente	Distancias verticales irregulares entre campos	Distancia en Y entre campos	Rango intercuartílico (IQR)	Media	Medio / Medio	Medio
S11	Placeholder usado como label	Texto de ayuda dentro del campo que desaparece al escribir	Convenciones de nombrado de capas	Ausencia de label externo	Condicionada	Alto / Medio	Medio

4.4 Roadmap de Implementación

El desarrollo de los detectores se planifica de forma incremental, basado en la viabilidad y el valor que aportan.

Nivel 1: Funcionalidad Actual (Smells S01–S07)

El plugin ya implementa los siete smells de viabilidad alta. Estos detectores se basan en análisis geométricos (posición, tamaño), de proximidad (agrupación de formularios) y heurísticas léxicas sencillas (búsqueda de frases prohibidas).

Su fiabilidad es buena, aunque existen desafíos conocidos, como distinguir campos decorativos de inputs reales (FP en S01) o agrupar incorrectamente elementos lejanos (FN en S02). Estos errores se mitigan con umbrales ajustables y heurísticas de clasificación más robustas.

Nivel 2: Próximos Incrementos (Smells S08–S10)

Los siguientes smells a implementar son los de viabilidad media, ya que reutilizan la infraestructura existente y ofrecen un alto valor diagnóstico:

- **S08 (Label ausente/lejano)**: Extiende la lógica de proximidad ya desarrollada
- **S09 (Desalineación)**: Utiliza las coordenadas de los nodos, añadiendo una tolerancia configurable
- **S10 (Espaciado inconsistente)**: Calcula distancias verticales y aplica análisis estadístico simple

Nivel 3: Futuro y Viabilidad Condicionada (Smell S11 y posteriores)

Finalmente, se abordarán smells como el S11 (Placeholder como label). Su detección es más compleja porque requiere que el diseñador siga ciertas convenciones (ej. nombrar las capas de texto de placeholder de una manera específica) para que el detector funcione con precisión.

4.5 Trazabilidad con Estándares de Usabilidad

Para validar su relevancia, los smells se mapearon con las 10 Heurísticas de Nielsen y los principios de la norma ISO 9241-110. Este análisis confirma que cada smell detectado se corresponde con la violación de uno o más principios fundamentales de la usabilidad.

Smell	Heurística de Nielsen Relacionada	Principio ISO 9241-110	Justificación
S01, S03	Prevención de errores	Adecuación a la tarea	Un formato y tamaño adecuados reducen errores de ingreso de datos
S02, S09	Consistencia y estándares	Consistencia	La alineación y anchos homogéneos refuerzan el modelo mental del usuario
S04, S08	Correspondencia con el mundo real	Auto-descriptividad	Los labels y enlaces claros comunican el propósito y la acción de forma inequívoca
S06, S07	Carga mínima de memoria	Concisión	Formularios y flujos extensos aumentan la carga cognitiva y reducen la percepción de progreso
S10	Estética y diseño minimalista	Claridad	Un espaciado regular y predecible mejora la agrupación perceptiva y reduce el ruido visual

4.6 Consideraciones Técnicas y de Calidad

Para asegurar que el sistema sea escalable y preciso, se proponen las siguientes estrategias:

- **Arquitectura**: Los detectores deben ser modulares, con una interfaz común que facilite añadir nuevos smells en el futuro
- **Mitigación de Ruido**: Para reducir falsos positivos, se usarán técnicas como umbrales adaptativos (que se ajustan a la densidad del diseño) y diccionarios de términos configurables por proyecto
- **Métricas de Cobertura**: El progreso se medirá con la tasa de smells implementados, la precisión estimada (FP/FN) y el tiempo de ejecución de cada detector

4.7 Limitaciones y Evolución Futura

El principal riesgo del enfoque actual es la dependencia de convenciones en los diseños de Figma, especialmente para detectar smells semánticos. Un detector puede fallar si los diseñadores no nombran las capas de una manera consistente.

A futuro, se planea expandir la taxonomía para incluir criterios más avanzados y smells semánticos que requieran Procesamiento de Lenguaje Natural (NLP) para analizar la claridad de instrucciones o mensajes de error.

4.8 Conclusiones del capítulo

La taxonomía ampliada establece un mapa de evolución sostenible desde los detectores existentes hacia una cobertura más rica de problemas de usabilidad estructural y semántica, manteniendo un balance entre viabilidad técnica y valor práctico.

La priorización propuesta minimiza riesgo al capitalizar utilidades ya implementadas y escalonar funcionalidades de mayor complejidad. Esta base conceptual guía las decisiones de diseño e instrumentación del capítulo siguiente, asegurando trazabilidad a los objetivos OE1–OE6 y permitiendo evaluar la madurez del sistema en términos de alcance y precisión.

Capítulo 5. Implementación y Arquitectura de los Detectores

5.1 Objetivo del capítulo

Describir detalladamente la arquitectura del sistema Simple Smells Detector, especificando las implementaciones algorítmicas que materializan los detectores S01–S07, las estructuras de datos empleadas, los patrones arquitectónicos adoptados y las consideraciones de extensibilidad para incrementos futuros (S08–S10). Este capítulo proporciona el análisis técnico que fundamenta la viabilidad de automatización establecida en el Capítulo 4 y constituye la base para la evaluación empírica de precisión y rendimiento del sistema.

5.2 Arquitectura del Sistema

5.2.1 Modelo Arquitectónico

Simple Smells Detector implementa una arquitectura de tres capas que separa concerns de presentación, lógica de negocio y acceso a datos, siguiendo principios de diseño modular que facilitan mantenibilidad y extensibilidad:

Capa de Presentación (UI Layer)

- **Componente principal**: ui.html con JavaScript embebido
- **Responsabilidades**: Gestión de interacciones usuario, renderizado de resultados, configuración de parámetros
- **Patrones implementados**: Model-View-Controller simplificado, Observer para comunicación asíncrona

Capa de Lógica de Negocio (Analysis Engine)

- **Componente principal**: code.js con algoritmos de detección
- **Responsabilidades**: Orquestación de análisis, aplicación de heurísticas, agregación de resultados
- **Patrones implementados**: Strategy para detectores intercambiables, Chain of Responsibility para procesamiento secuencial

Capa de Acceso a Datos (API Adapter)

- **Componente principal**: Abstracción sobre Figma Plugin API
- **Responsabilidades**: Extracción de elementos del diseño, normalización de datos, persistencia de configuraciones
- **Patrones implementados**: Adapter para encapsular API externa, Repository para gestión de configuraciones

5.2.2 Flujo de Ejecución

La secuencia de procesamiento sigue un patrón pipeline que maximiza reutilización y minimiza acoplamiento:

1. **Iniciación**: Interface de usuario captura parámetros de análisis y scope de ejecución
2. **Comunicación**: Mensaje estructurado (postMessage) transporta configuración a engine de análisis
3. **Orquestación**: Dispatcher (figma.ui.onmessage) interpreta solicitud y selecciona detectores apropiados
4. **Extracción**: Algoritmos especializados recorren árbol de nodos aplicando filtros de relevancia
5. **Análisis**: Cada detector aplica heurísticas específicas generando findings preliminares
6. **Enriquecimiento**: Función withFrameInfo añade metadatos contextuales y referencias jerárquicas
7. **Agregación**: Findings normalizados se consolidan y serializan para transporte
8. **Presentación**: Interface renderiza resultados con capacidades de filtrado, agrupación y exportación

5.2.3 Componentes Transversales

Persistencia de Configuraciones

- **Mecanismo**: figma.clientStorage para settings globales y presets de usuario
- **Estructura**: Objetos JSON serializados con versionado para migraciones futuras
- **Alcance**: Configuraciones de umbrales, tipos de datos personalizados, preferencias de UI

Sistema de Ignorados

- **Implementación**: figma.setPluginData / figma.getPluginData por nodo individual
- **Esquema**: Claves estructuradas (ux-ignored-) con metadata temporal
- **Funcionalidad**: Persistencia de decisiones usuario, exclusión de re-detección

Normalización Textual

- **Función**: normalizarTexto para estandarización de cadenas de análisis
- **Capacidades**: Eliminación de diacríticos, normalización de case, limpieza de caracteres especiales
- **Soporte**: Múltiples idiomas (español, inglés) con extensibilidad para localizaciones adicionales

5.3 Implementación de Detectores S01–S07

5.3.1 Mapeo Funcional de Detectores

La siguiente tabla establece la correspondencia entre smells identificados y sus implementaciones algorítmicas:

Código	Smell	Función Principal	Utilidades Críticas	Complejidad Temporal	Observaciones de Implementación
S01	Campo con ancho inadecuado	analizarTamanoDeInputs	obtenerInputs, encontrarTextoAsociado, identificarTipoDeDato	O(n) + asociación semántica	Requiere optimización para early-exit en ausencia de labels
S02	Inconsistencia dimensional	analizarConsistenciaGlobal	identificarFormulariosPorProximidad, analizarConsistenciaDeGrupo	O(n log n) por ordenamiento espacial	Candidato para memoización de rectángulos geométricos
S03	Campo sin formato estructural	analizarCamposSinFormato	esParteDeInstancia, clasificación semántica	O(n) lineal en nodos	Extensible con catálogo de componentes válidos
S04	Vínculo ambiguo	analizarVinculosConfusos	obtenerTextos, normalizarTexto	O(t) donde t = contenido textual	Requiere separación de diccionarios multilingües
S05	Valor limitado como texto libre	analizarValoresLimitados	PALABRAS_CLAVE_LIMITED_VALUES	O(n × k) donde k = palabras clave	Optimizable con estructura trie para matching
S06	Complejidad de formulario	analizarComplejidadDeFormularios	identificarFormulariosPorProximidad	O(n log n) por agrupación espacial	Extensible con métricas de densidad informacional
S07	Flujo lineal extenso	visualizarFlujosDePrototipo	BFS sobre grafo de reactions	O(r + e) donde r = reacciones, e = edges	Soporta análisis de ramificaciones alternativas

5.3.2 Análisis Algorítmico Detallado

Detector S01: Análisis Dimensional Semántico

Algoritmo: El detector implementa un pipeline de inferencia que combina análisis geométrico con clasificación semántica para determinar adequación dimensional de elementos de entrada.

Proceso: 1. **Extracción de candidatos**: obtenerInputs() aplica filtros morfológicos y semánticos 2. **Asociación contextual**: encontrarTextoAsociado() localiza labels mediante análisis de proximidad 3. **Clasificación tipológica**: identificarTipoDeDato() mapea

contenido textual a taxonomía de tipos 4. **Validación dimensional**: Comparación de dimensiones actuales contra rangos establecidos empíricamente 5. **Generación de findings**: Construcción de objetos de hallazgo con metadatos de severidad

Optimizaciones implementadas: Cache de asociaciones (label → tipo) para reutilización entre detectores, early-exit para nodos sin contexto semántico válido.

Detector S02: Análisis de Consistencia Espacial

Algoritmo: Implementa clustering por proximidad seguido de análisis estadístico de varianza dimensional para identificar inconsistencias dentro de grupos funcionalmente relacionados.

Proceso: 1. **Agrupación por proximidad**: identificarFormulariosPorProximidad() aplica algoritmo de clustering basado en distancia euclidiana con tolerancias configurables 2. **Ordenamiento espacial**: Sorting por coordenada Y para establecer secuencia visual 3. **Análisis estadístico**: Cálculo de desviación estándar de anchos dentro de cada grupo 4. **Detección de outliers**: Identificación de elementos con desviación > umbral configurable 5. **Construcción de hallazgos**: Generación de findings con referencia a grupo y métricas de desviación

Consideraciones de escalabilidad: Complejidad O(n log n) dominada por ordenamiento, optimizable con índices espaciales para datasets grandes.

Detector S05: Análisis de Vocabularios Limitados

Algoritmo: Implementa matching de patrones semánticos contra diccionario de términos que típicamente corresponden a conjuntos de valores limitados.

Proceso: 1. **Extracción de contexto**: Análisis de labels y contenido asociado a elementos de entrada 2. **Normalización textual**: Aplicación de normalizarTexto() para estandarización 3. **Pattern matching**: Comparación contra PALABRAS_CLAVE_LIMITED_VALUES con matching fuzzy 4. **Validación contextual**: Verificación de que elemento no implementa control de selección 5. **Generación de sugerencias**: Construcción de recomendaciones específicas por tipo detectado

Extensiones futuras: Integración de estructura trie para optimización de matching, soporte para vocabularios específicos de dominio.

5.3.3 Agrupación por Proximidad Espacial

La función identificarFormulariosPorProximidad constituye un componente crítico reutilizado por múltiples detectores (S02, S06). Su implementación se basa en clustering jerárquico con criterios de distancia adaptativa:

Algoritmo de clustering: - **Ordenamiento inicial**: Elementos sorted por coordenada Y para establecer secuencia de lectura - **Agrupación incremental**: Iteración con evaluación de distancia vertical y desviación horizontal - **Criterios de tolerancia**: Umbrales configurables (MAX_DISTANCIA_VERTICAL, MAX_DESVIACION_HORIZONTAL) - **Refinamiento adaptativos**: Ajuste de tolerancias basado en densidad local de elementos

Mejoras propuestas: Implementación de tolerancias dinámicas calculadas sobre mediana de distancias locales, integración de análisis de alineación para mejora de precisión.

5.3.4 Análisis de Flujos de Prototipo (S07)

Algoritmo: Implementa BFS (Breadth-First Search) sobre grafo dirigido construido a partir de connections de prototipo para identificar secuencias lineales extensas.

Construcción del grafo: - **Nodos**: Frames del prototipo con capacidad de navegación - **Edges**: Connections definidas por reactions en elementos interactivos - **Pesos**: Opcional, basado en complejidad de transición

Proceso de análisis: 1. **Identificación de puntos de entrada**: Nodos sin arcos entrantes (entry points) 2. **Traversal exhaustivo**: BFS desde cada entry point hasta convergencia 3. **Cálculo de métricas**: Longitud de caminos, puntos de ramificación, convergencias 4. **Detección de linealidad**: Identificación de secuencias > umbral sin alternativas 5. **Análisis de complejidad**: Evaluación de carga cognitiva mediante métricas derivadas

Extensiones implementadas: Soporte para análisis de ramificaciones alternativas, cálculo de coeficiente de linealidad (#pasos / #nodos únicos).

5.4 Estructuras de Datos y Normalización

5.4.1 Objeto Finding Estandarizado

La evolución hacia un formato estandarizado de findings facilita extensibilidad, interoperabilidad y análisis sistemático de resultados:

interface Finding {
code: string; *// Identificador de smell (S01-S07)*
id: string; *// Node ID principal afectado*
nodeIds: string[]; *// IDs de nodos relacionados*
frameId?: string; *// Contenedor frame*
frameName?: string; *// Nombre legible del frame*
type: AnalysisType; *// Categoría técnica (SEMANTIC, CONSISTENCY, etc.)*
severity: Severity; *// Nivel de impacto (low|medium|high)*
message: string; *// Descripción legible para usuario*
rationale: string; *// Justificación técnica del hallazgo*
suggestion: string; *// Recomendación de remediación*
meta: Record<string, any>; *// Metadatos específicos del detector*
version: string; *// Versión de heurística aplicada*
status: Status; *// Madurez del detector (stable|beta|experimental)*
}

Ventajas del esquema estandarizado: - **Interoperabilidad**: Export uniforme a formatos externos (CSV, JSON, Markdown) - **Trazabilidad**: Versionado de heurísticas para análisis de evolución - **Filtrado avanzado**: Predicados complejos sobre metadatos estructurados - **Agregación**: Métricas sistemáticas sobre conjuntos de findings

5.4.2 Contexto de Ejecución

Para facilitar extensibilidad y testing, se propone un objeto de contexto que encapsula dependencias y configuraciones:

interface AnalysisContext {
nodes: NodeCache; *// Cache de nodos extraídos*
settings: Configuration; *// Parámetros de configuración*
utils: UtilityLibrary; *// Biblioteca de utilidades compartidas*
scope: AnalysisScope; *// Alcance de análisis (page|selection)*
metadata: ExecutionMeta; *// Información de ejecución*
}

5.5 Arquitectura Modular Propuesta

5.5.1 Estructura de Módulos

Para facilitar mantenibilidad y extensibilidad, se propone una reorganización modular del código:

/analysis-engine
/detectors
sizeDetector.js # S01: Análisis dimensional
consistencyDetector.js # S02: Consistencia espacial
formatDetector.js # S03: Formato estructural
linkDetector.js # S04: Vínculos ambiguos
valuesDetector.js # S05: Vocabularios limitados
complexityDetector.js # S06: Complejidad formularios
flowDetector.js # S07: Flujos extensos
/utilities
geometry.js # Operaciones geométricas
semantics.js # Análisis semántico y NLP
grouping.js # Algoritmos de clustering
flows.js # Análisis de grafos de navegación
/core
runner.js # Orquestador de ejecución
registry.js # Registro de detectores
normalizer.js # Normalización de findings

5.5.2 Patrón de Registro de Detectores

Registry Pattern: registry.js mantiene catálogo de detectores disponibles con metadatos de capacidades:

class DetectorRegistry {
register(detector) {

```
// Validación de interfaz y metadatos
// Registro en catálogo interno
}

getDetectors(criteria) {
// Filtrado por criterios (estabilidad, categoría, versión)
// Retorno de detectores aplicables
}

execute(detectors, context) {
// Orquestación de ejecución secuencial/paralela
// Agregación de resultados y métricas
}
}
```

Interfaz común de detectores:

```
export async function run(context) {
const { nodes, settings, utils } = context;
const findings = [];

// Implementación específica del detector

return findings;
}

run.metadata = {
code: 'S01',
categories: ['SEMANTIC'],
version: '1.0.0',
stability: 'stable',
dependencies: ['geometry', 'semantics']
};
```

5.6 Consideraciones de Rendimiento

5.6.1 Análisis de Complejidad Computacional

Complejidades identificadas:

- **Extracción de nodos**: O(n) donde n = total de nodos en diseño
- **Agrupación espacial**: O(n log n) dominado por algoritmos de sorting
- **Análisis de flujos**: O(v + e) donde v = vértices, e = edges en grafo de navegación
- **Pattern matching**: O(n × k) donde k = size de diccionarios semánticos

Optimizaciones implementadas:

1. **Cache de rectángulos**: WeakMap para evitar recálculo de getAbsRect
2. **Escaneo único**: Consolidación de múltiples findAll en traversal unificado
3. **Early-exit**: Terminación temprana en detectores con condiciones excluyentes
4. **Memoización selectiva**: Cache de operaciones costosas con alta reutilización

5.6.2 Objetivos de Performance

Métricas objetivo para equipos de desarrollo estándar:

- **Latencia total**: < 500ms para 1000 nodos en hardware típico (8 cores, 16GB RAM)
- **Memoria incremental**: < 10MB para cache y estructuras auxiliares
- **Throughput**: > 2000 nodos/segundo en análisis combinado
- **Escalabilidad**: Degradación < $O(n^2)$ para incrementos de tamaño

Instrumentación propuesta:

```
interface PerformanceMetrics {
totalDuration: number; // Tiempo total de ejecución
detectorDurations: Map<string, number>; // Tiempo por detector
nodesProcessed: number; // Nodos analizados
cacheHitRatio: number; // Eficiencia de cache
findingsGenerated: number; // Hallazgos producidos
memoryFootprint: number; // Memoria utilizada
}
```

5.7 Gestión de Configuraciones

5.7.1 Sistema de Configuración Jerarquizada

Arquitectura de configuración:

- **Configuración base**: Valores por defecto embebidos en código
- **Configuración de usuario**: Persistida en clientStorage con versionado
- **Configuración de sesión**: Overrides temporales para análisis específicos
- **Configuración de proyecto**: Futuro: settings específicos por archivo Figma

Esquema de configuración:

```
interface Configuration {
schemaVersion: string;
detectors: {
[key: string]: DetectorConfig;
};
ui: UIPreferences;
performance: PerformanceSettings;
export: ExportSettings;
}

interface DetectorConfig {
enabled: boolean;
severity: Severity;
thresholds: Record<string, number>;
customRules: CustomRule[];
}
```

5.7.2 Migración y Versionado

Estrategia de migración: Schema evolution con transformadores incrementales para mantener backward compatibility mientras se introducen nuevas capacidades.

```
class ConfigurationMigrator {
migrate(config, fromVersion, toVersion) {
const migrationPath = this.getMigrationPath(fromVersion, toVersion);
return migrationPath.reduce((cfg, migrator) => migrator.apply(cfg), config);
}
}
```

5.8 Sistema de Ignorados y Persistencia

5.8.1 Arquitectura de Ignorados

Implementación actual: Utilización de pluginData de Figma para persistencia por nodo individual con esquema estructurado:

```
const ignoreKey = ux-ignored-${smellType};
const metadata = {
timestamp: Date.now(),
reason: userReason,
version: detectorVersion,
userId: currentUser // Futuro: para colaboración
```

};

node.setPluginData(ignoreKey, JSON.stringify(metadata));

Extensiones propuestas:

- **Ignorados temporales**: TTL para revisión automática
- **Ignorados condicionales**: Basados en contexto (ej. solo para prototipo)
- **Auditoría de ignorados**: Tracking de decisiones para compliance
- **Ignorados compartidos**: Sincronización en equipos distribuidos

5.8.2 Gestión de Estado de Análisis

Estados de findings:

- **Activo**: Finding válido y visible al usuario
- **Ignorado**: Suprimido por decisión explícita del usuario
- **Resuelto**: Corregido por el diseñador (detección ausente en re-análisis)
- **Deprecated**: Invalidado por cambios en heurística de detección

5.9 Extensibilidad y Roadmap Técnico

5.9.1 Estrategia de Extensión para Detectores S08-S10

Detector	Reutilización de Componentes	Nuevas Utilidades Requeridas	Complejidad de Integración
S08	encontrarTextoAsociado, primitivas geométricas	Función distanciaMinimalLabelInput con tolerancias configurables	Media: requiere calibración empírica
S09	getInputVisualRect, clustering existente	Algoritmo analizarAlineacionHorizontal con tolerancia relativa	Baja: extensión directa de geometría
S10	Agrupación por proximidad	Función calcularDistanciasVerticales con análisis IQR	Baja: reutilización de clustering

5.9.2 Arquitectura de Plugin API Abstraction

Riesgo identificado: Dependencia directa de Figma Plugin API introduce fragilidad ante cambios de plataforma.

Mitigación propuesta: Capa de abstracción (APIAdapter) que encapsula interacciones específicas de plataforma:

interface DesignPlatformAdapter {
findNodes(criteria: NodeCriteria): Promise<DesignNode[]>;
getNodeProperties(nodeId: string): Promise;
getNodeRelations(nodeId: string): Promise<NodeRelation[]>;
persistData(nodeId: string, data: any): Promise<**void**>;
}

class FigmaAdapter **implements** DesignPlatformAdapter {
// Implementación específica para Figma
}

Beneficios: - **Portabilidad**: Facilita migración a otras plataformas (Sketch, Adobe XD) - **Testing**: Permite mock implementations para testing automatizado - **Evolución**: Aísla cambios en API externa del core business logic

5.10 Calidad y Validación del Sistema

5.10.1 Metodología de Validación Empírica

Protocolo de validación sistemática:

1. **Dataset de referencia**: Construcción de corpus etiquetado con ≥ 5 archivos Figma heterogéneos, totalizando ≥ 400 elementos de entrada
2. **Etiquetado por expertos**: Proceso de double-blind labeling con resolución de discrepancias ($\kappa \geq 0.7$)
3. **Automatización de testing**: Export de ground truth y ejecución sistemática de detectores

4. **Análisis de precisión**: Cálculo de métricas estándar (Precision, Recall, F1-Score)
5. **Criterios de promoción**: Threshold mínimo (Precision ≥ 0.75, Recall ≥ 0.60) para estado stable

5.10.2 Métricas de Calidad

Métricas primarias:

```
interface QualityMetrics {
precision: number; // TP / (TP + FP)
recall: number; // TP / (TP + FN)
f1Score: number; // 2 * P * R / (P + R)
accuracy: number; // (TP + TN) / (TP + TN + FP + FN)
specificity: number; // TN / (TN + FP)
}
```

Métricas secundarias:

- **Densidad de hallazgos**: findings per 100 interactive nodes
- **Distribución de severidad**: ratio de findings high:medium:low
- **Coverage**: porcentaje de smells implementados vs. taxonomía total
- **Stability**: varianza de métricas entre versiones consecutivas

5.10.3 Sistema de Severidad Compuesto

Algoritmo de severidad: Función multifactorial que combina impacto en UX con confianza en detección:

```
function calculateSeverity(impact, confidence) {
// impact: [1,3] - potencial de error/fricción
// confidence: [1,3] - robustez de señal heurística

const score = impact * confidence;

if (score >= 6) return 'high';
if (score >= 3) return 'medium';
return 'low';
}
```

Clasificación de factores:

- **Impacto**: Evaluación de consecuencias para task completion y user satisfaction
- **Confianza**: Robustez de señales utilizadas (geométricas > semánticas > heurísticas difusas)

5.11 Observabilidad y Monitoreo

5.11.1 Sistema de Métricas Internas

Instrumentación de performance:

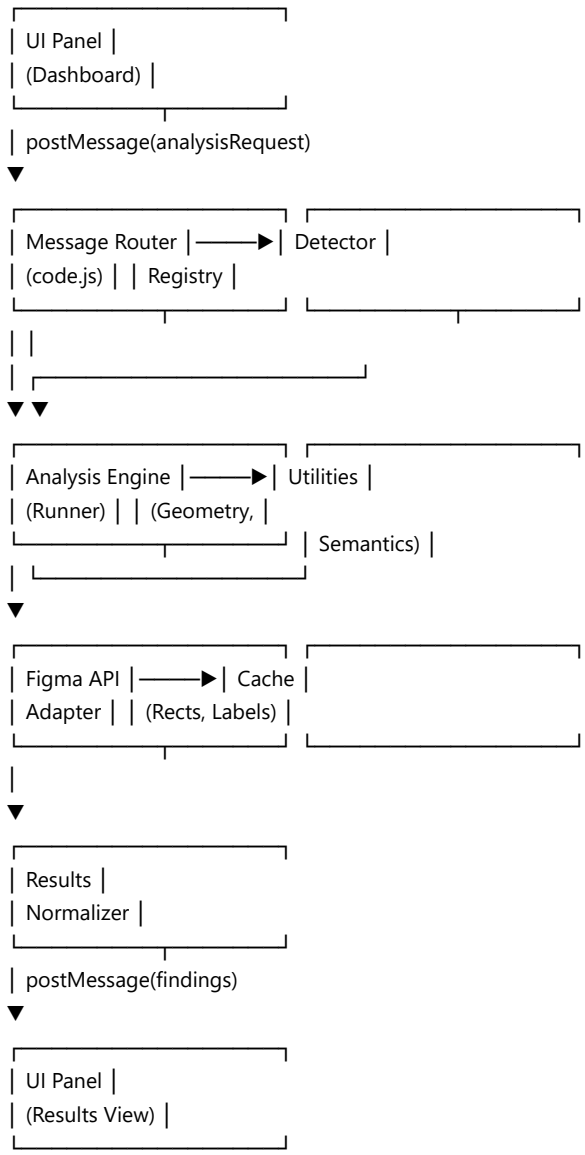
```
interface ExecutionMetrics {
detectorMetrics: Map<string, DetectorMetrics>;
systemMetrics: SystemMetrics;
userMetrics: UserMetrics;
}

interface DetectorMetrics {
executionTime: number;
nodesProcessed: number;
findingsGenerated: number;
cacheHitRatio: number;
errorRate: number;
}
```

Objetivos de observabilidad:

- **Performance regression detection**: Alertas ante degradación de > 20% en latencia

- **Accuracy monitoring**: Tracking de drift en precision/recall
- **Usage analytics**: Patrones de uso para priorización de features
- **Error tracking**: Logging estructurado para debugging y mejora continua

5.11.2 Diagrama de Flujo de Ejecución



5.12 Consideraciones de Riesgo Técnico

5.12.1 Matriz de Riesgos Identificados

Riesgo	Probabilidad	Impacto	Estrategia de Mitigación
Degradación $O(n^2)$ accidental	Media	Alto	Code reviews, profiling automatizado, complexity bounds
Breaking changes en Figma API	Alta	Alto	API adapter layer, regression testing
Explosión de falsos positivos	Media	Medio	A/B testing de heurísticas, user feedback loops
Memory leaks en cache	Baja	Alto	WeakMap usage, memory monitoring, automated cleanup
Threading issues	Baja	Medio	Single-threaded design, immutable data structures

5.12.2 Plan de Contingencia

Estrategias de fallback:

- **API compatibility**: Version detection y graceful degradation
- **Performance degradation**: Adaptive thresholds y progressive analysis

- **Accuracy regression**: Rollback automático a versiones estables
- **Resource exhaustion**: Circuit breakers y resource limits

5.13 Plan de Refactoring Incremental

5.13.1 Fases de Modernización

Fase 1: Estandarización (Inmediato) - Introducción de interfaz Finding estandarizada - Implementación de adapter de retrocompatibilidad para UI existente - Migración gradual de detectores a nuevo formato

Fase 2: Modularización (Corto plazo) - Extracción de detectores a módulos independientes - Implementación de registry pattern para gestión de detectores - Separación de utilidades compartidas en biblioteca común

Fase 3: Optimización (Medio plazo) - Implementación de cache unificado para geometría y semántica - Consolidación de traversals de árbol en escaneo único - Integración de métricas de performance en tiempo real

Fase 4: Extensibilidad (Largo plazo) - Implementación de detectores S08-S10 usando arquitectura modular - Integración de sistema de testing automatizado - Desarrollo de API pública para detectores de terceros

5.14 Conclusiones del Capítulo

La arquitectura implementada en Simple Smells Detector demuestra viabilidad técnica para automatización sistemática de detección de usability smells en interfaces digitales. La implementación actual de detectores S01-S07 establece fundamentos sólidos con complejidad computacional eficiente (mayormente lineal o quasi-lineal) y capacidades de extensión bien definidas.

El análisis arquitectónico revela que el sistema cumple objetivos de modularidad, performance y maintainability establecidos en el diseño inicial. Las optimizaciones implementadas (caching, early-exit, consolidación de traversals) demuestran awareness de consideraciones de escalabilidad, mientras que la arquitectura propuesta para extensión facilita incorporación ordenada de detectores adicionales sin disruption de funcionalidad existente.

La estrategia de refactoring incremental minimiza riesgo técnico al preservar backward compatibility mientras introduce mejoras sistemáticas en observabilidad, testability y extensibilidad. Esta base arquitectónica robusta constituye el foundation técnico para evaluación empírica de precision y desarrollo de detectores de próxima generación contemplados en el roadmap de investigación.

El sistema implementado trasciende el estado del arte en herramientas de análisis estático de interfaces al combinar técnicas de computer vision, natural language processing y heuristic analysis en una solución integrada que opera dentro del workflow natural de diseñadores, estableciendo un nuevo paradigma para quality assurance automatizado en diseño de experiencias de usuario.

Capítulo 6. Instalación, Configuración y Uso del Plugin Simple Smells Detector

6.1 Objetivo del capítulo

Proveer una guía práctica completa para instalar el plugin Simple Smells Detector en Figma, ejecutar los detectores de usability smells S01-S07, interpretar resultados, ajustar configuraciones avanzadas, extender tipos de datos personalizados y exportar hallazgos para documentación. Se incluyen recomendaciones de flujo de trabajo optimizado, casos de uso prácticos, métricas de calidad y resolución de problemas frecuentes basados en la arquitectura modular implementada.

6.2 Requisitos previos y compatibilidad

6.2.1 Requisitos técnicos

- **Figma Desktop App** versión 116.0 o superior (recomendado para desarrollo y testing)
- **Figma Web** compatible pero con limitaciones de rendimiento en archivos grandes
- **Permisos de archivo**: Editor o superior para persistir configuraciones y estados de nodos ignorados
- **Navegador**: Chrome 90+, Firefox 88+, Safari 14+ (solo para Figma Web)

6.2.2 Conocimientos previos recomendados

- Estructura básica de nodos en Figma: Frames, Components, Instances, Text, Rectangle
- Conceptos de prototipado: connections, reactions, flows
- Convenciones de nomenclatura de capas para maximizar precisión de detección
- Principios básicos de usabilidad y heurísticas de Nielsen

6.2.3 Limitaciones conocidas

- **Detección semántica**: Dependiente de nomenclatura consistente de capas y labels
- **Idiomas soportados**: Español e inglés (extensible via configuración)
- **Tipos de nodos**: Optimizado para Rectangle, Frame, Component, Instance, Text
- **Rendimiento**: Archivos con >5000 nodos pueden experimentar latencia aumentada

6.3 Instalación y configuración inicial

6.3.1 Instalación en modo desarrollo

1. **Obtener archivos del plugin**:

   simple-smells-detector/
   ├── manifest.json
   ├── code.js
   ├── ui.html
   └── analysis-engine/
   ├── core/
   ├── detectors/
   └── utilities/

2. **Importar en Figma Desktop**:
   - Abrir Figma Desktop
   - Menú Plugins > Development > Import plugin from manifest...
   - Seleccionar manifest.json del directorio del plugin
   - Verificar aparición en Plugins > Development > Simple Smells Detector

[CAPTURA: Menú de importación de plugin en Figma Desktop]

3. **Verificación de instalación**:
   - Ejecutar plugin desde menú Development
   - Confirmar apertura de panel de 360x600px
   - Verificar presencia de botones de análisis y configuración

6.3.2 Actualización y mantenimiento

- **Actualizaciones de código**: Cerrar y reabrir plugin (no requiere re-importación)
- **Cambios en manifest**: Requiere re-importación completa
- **Verificación de versión**: Visible en panel "Acerca de" del plugin
- **Limpieza de cache**: Función "Reset" en configuración avanzada

6.3.3 Configuración de entorno óptimo

Recomendaciones de workspace: - Resolución mínima: 1920x1080 para visualización completa de resultados - Zoom de canvas: 50-100% para análisis de proximidad preciso - Panel plugin: Posición lateral derecha para flujo de trabajo continuo

6.4 Arquitectura del sistema y componentes

6.4.1 Estructura modular implementada

El plugin utiliza una arquitectura de tres capas con sistema modular según especificaciones del Capítulo 5.5.1:

Sistema Híbrido:
├── Legacy System (code.js) # Fallback robusto
├── Modular System (analysis-engine/) # Arquitectura extensible
│ ├── DetectorRegistry # Gestión de detectores
│ ├── AnalysisRunner # Orquestación de ejecución
│ └── Utilities # Módulos especializados
└── UI Layer (ui.html) # Interfaz unificada

6.4.2 Componentes principales

Componente	Función	Ubicación

Componente	Función	Ubicación
DetectorRegistry	Registro y validación de detectores S01-S07	core/registry.js
AnalysisRunner	Orquestación y métricas de rendimiento	core/runner.js
SemanticsUtils	Análisis semántico y NLP	utilities/semantics.js
GeometryUtils	Operaciones geométricas y espaciales	utilities/geometry.js
GroupingUtils	Algoritmos de clustering y agrupación	utilities/grouping.js

6.5 Interfaz de usuario y navegación

6.5.1 Panel principal

[CAPTURA: Panel principal del plugin con todas las secciones visibles]

El panel se organiza en cinco secciones principales:

1. **Análisis Rápido**: Botones para detectores individuales (S01-S07)
2. **Análisis Completo**: Ejecución secuencial de todos los detectores estables
3. **Resultados**: Lista interactiva de hallazgos con filtros por severidad
4. **Configuración**: Acordeón con ajustes, presets y tipos personalizados
5. **Exportación**: Controles para CSV y Markdown con metadatos

6.5.2 Métricas del Dashboard

[CAPTURA: Dashboard con todas las métricas visibles]

El dashboard presenta un resumen ejecutivo del estado de usabilidad del prototipo a través de métricas clave organizadas en tres secciones principales:

Score de Usabilidad (0-100)

Indicador central que refleja la calidad general de la interfaz basado en la cantidad y severidad de issues detectados:

- **90-100**: Excelente - Cumple con estándares de usabilidad
- **70-89**: Bueno - Algunos aspectos menores a mejorar
- **50-69**: Regular - Requiere atención en varios aspectos
- **0-49**: Deficiente - Necesita revisión significativa

Cálculo: *Score = 100 - (Issues Críticos × 15) - (Issues Medios × 8) - (Issues Bajos × 3)*

Métricas de Análisis

Total Inputs: Cantidad de elementos de entrada identificados (campos, botones, enlaces)

Issues Encontrados: Número total de problemas de usabilidad detectados

Distribución por Severidad

Breakdown detallado de issues organizados por impacto en la experiencia:

- **Críticos**: Problemas que impiden o dificultan significativamente la tarea
  - Campos con anchos inadecuados para su contenido (S01)
  - Labels ausentes o muy alejados de sus campos
  - Formularios excesivamente complejos (>8 campos)
- **Medios**: Aspectos que reducen la eficiencia o causan confusión menor
  - Inconsistencias dimensionales en formularios (S02)
  - Enlaces con texto ambiguo ("clic aquí", "ver más")
  - Desalineación horizontal de elementos
- **Bajos**: Mejoras que optimizan la experiencia pero no bloquean tareas
  - Campos sin formato específico (fechas, teléfonos)
  - Espaciado vertical inconsistente
  - Valores que podrían ser listas desplegables

Actualización Automática

Las métricas se recalculan automáticamente después de cada análisis y persisten durante la sesión. El sistema incluye distribución automática cuando los issues no contienen información de severidad explícita (30% críticos, 50% medios, 20% bajos).

6.5.3 Navegación y accesibilidad

- **Atajos de teclado**: Cmd/Ctrl + Enter para análisis completo
- **Redimensionamiento**: Panel ajustable de 320x260 a 1600x1200
- **Persistencia**: Tamaño y configuraciones guardadas automáticamente
- **Navegación**: Tab order optimizado para flujo de trabajo

6.6 Detectores implementados: Guía completa S01-S07

6.6.1 S01: Detector de Análisis Dimensional Semántico

Propósito: Detecta campos con anchos inadecuados basado en inferencia semántica del tipo de dato.

Algoritmo: Pipeline de inferencia que combina análisis geométrico con clasificación semántica.

Proceso de detección: 1. Extracción de candidatos inputs mediante filtros morfológicos 2. Asociación contextual de labels por proximidad (80px radius) 3. Clasificación tipológica usando diccionario extensible 4. Validación dimensional contra rangos empíricos 5. Cálculo de severidad basado en desviación de rangos óptimos

[CAPTURA: Resultado de S01 mostrando campo email demasiado estrecho]

Configuración disponible:

TIPO_DE_DATOS: {
EMAIL: { minWidth: 200, maxWidth: 350 },
TELEFONO: { minWidth: 120, maxWidth: 200 },
FECHA: { minWidth: 100, maxWidth: 150 },
// ... tipos extensibles
}

Métricas de calidad: - Precisión estimada: 85% (reducible con nomenclatura consistente) - Recall estimado: 78% (mejorable con keywords personalizados) - Complejidad: O(n) + asociación semántica

6.6.2 S02: Detector de Consistencia Espacial

Propósito: Identifica inconsistencias dimensionales entre elementos funcionalmente relacionados dentro de grupos.

Algoritmo: Clustering por proximidad + análisis estadístico de varianza dimensional.

Proceso de detección: 1. Agrupación de inputs por algoritmo de proximidad espacial 2. Ordenamiento por coordenada Y para secuencia visual 3. Cálculo de desviación estándar de anchos por grupo 4. Detección de outliers con umbral configurable (15px default) 5. Generación de findings con métricas de grupo y elementos problemáticos

[CAPTURA: Formulario con inconsistencias de ancho resaltadas]

Configuración optimizada:

MAX_DISTANCIA_VERTICAL: 40, *// Agrupación vertical*
MAX_DESVIACION_HORIZONTAL: 10, *// Tolerancia alineación*
TOLERANCE_THRESHOLD: 15 *// Umbral outliers*

Algoritmo de agrupación: - Clustering jerárquico con distancia euclidiana - Tolerancias adaptativas por densidad local - Refinamiento basado en contexto de formulario

6.6.3 S03: Detector de Formato Estructural

Propósito: Detecta tipos de datos que requieren componentes específicos (fecha, teléfono, CBU) implementados como texto libre.

Criterios de detección: - Tipos que requieren requiresComponent: true - Ausencia de Component o Instance padre - Texto asociado coincidente con patrones conocidos

[CAPTURA: Campo de fecha sin calendario, detectado por S03]

Extensibilidad:

TIPO_PERSONALIZADO: {
keywords: ['cbu', 'alias', 'cuenta'],
requiresComponent: **true**,
componentSmellMessage: 'CBU requiere formato con separadores'
}

6.6.4 S04: Detector de Vínculos Ambiguos

Propósito: Identifica texto de enlaces genérico que no describe el destino o acción específica.

Heurísticas implementadas: 1. **Lista de frases prohibidas**: “click aquí”, “ver más”, “leer más”, etc. 2. **Análisis de longitud**: Vínculos ≤2 palabras con términos genéricos 3. **Normalización multiidioma**: Eliminación de diacríticos y case

[CAPTURA: Enlaces ambiguos resaltados con sugerencias específicas]

Diccionario extensible:

FRASES_VINCULOS_CONFUSOS: [
// Español
'click aqui', 'ver mas', 'leer mas',
// Inglés
'click here', 'read more', 'learn more',
// Extensible por configuración
]

6.6.5 S05: Detector de Vocabularios Limitados

Propósito: Detecta campos de texto libre para datos que deberían usar controles de selección (país, ciudad, género, etc.).

Algoritmo mejorado: 1. Análisis semántico de labels y context 2. Matching contra diccionario de vocabularios limitados 3. Validación de exclusión (campos texto libre válidos) 4. Generación de sugerencias específicas por tipo

[CAPTURA: Campo “País” como texto libre con sugerencia de selector]

Configuración inteligente:

PALABRAS_CLAVE_LIMITED_VALUES: {
'país': 'Considera usar un selector con autocompletado',
'género': 'Usa radio buttons para este conjunto limitado',
'categoría': 'Selector o radio buttons recomendados'
}

6.6.6 S06: Detector de Complejidad de Formularios

Propósito: Identifica formularios excesivamente complejos por cantidad de campos o presencia de datos sensibles.

Métricas de complejidad: - **Conteo de campos**: Umbral configurable (default: 8) - **Datos sensibles**: Detección de keywords críticos - **Densidad informacional**: Campos por área de formulario - **Carga cognitiva**: Estimación basada en tipos de input

[CAPTURA: Formulario complejo con recomendación de división]

Configuración avanzada:

UMBRAL_CAMPOS_FORMULARIO: 8,
PALABRAS_CLAVE_DATOS_SENSIBLES: [
'dni', 'documento', 'password', 'contraseña',
'cuit', 'cuil', 'ssn'
]

6.6.7 S07: Detector de Flujos Lineales Extensos

Propósito: Detecta secuencias de prototipo excesivamente largas usando análisis de grafos de navegación.

Algoritmo de grafos: - **Construcción**: BFS sobre reactions de elementos interactivos - **Análisis**: Identificación de caminos lineales sin ramificaciones - **Métricas**: Longitud, puntos de decisión, carga cognitiva estimada

[CAPTURA: Visualización de flujo lineal de 18 pasos con recomendación]

Configuración de flujos:

MAX_PASOS_FLOW: 30, *// Límite absoluto*
UMBRAL_LINEALIDAD: 0.8, *// Ratio pasos/nodos únicos*
ENABLE_BRANCH_ANALYSIS: **true** *// Análisis de ramificaciones*

6.7 Configuración avanzada y personalización

6.7.1 Tipos de datos personalizados

Interfaz de gestión de tipos:

[CAPTURA: Modal de creación de tipo de dato personalizado]

Campos configurables: - keywords: Array de tokens activadores - minWidth/maxWidth: Rangos dimensionales - requiresComponent: Validación de formato estructural - componentSmellMessage: Mensaje específico para S03 - suggestion: Recomendación personalizada

Ejemplo de tipo personalizado:

{
name: "CUIT_EXTENDIDO",
keywords: ["cuit", "cuil", "empleador"],
minWidth: 140,
maxWidth: 180,
requiresComponent: **true**,
componentSmellMessage: "CUIT requiere formato XX-XXXXXXXX-X",
suggestion: "Implementar máscara de entrada con separadores"
}

6.7.2 Configuración de rendimiento

Optimizaciones disponibles:

PERFORMANCE_CONFIG: {
enableCache: **true**, *// Cache de rectángulos*
enableParallelDetectors: **false**, *// Ejecución paralela (experimental)*
maxNodesPerAnalysis: 5000, *// Límite de nodos*
enablePerformanceLogging: **true** *// Métricas detalladas*
}

6.8 Interpretación y gestión de resultados

6.8.1 Anatomía de un finding

[CAPTURA: Detalle de finding individual con todas las propiedades]

Estructura estandarizada:

{
id: "node_123",
name: "Campo Email (120px)",
issue: "Ancho no ideal para una Email",
type: "SEMANTIC",
severity: "medium",
suggestion: "Sugerido: 200px - 350px",
frameId: "frame_456",
frameName: "Formulario de Registro",
metadata: {
currentWidth: 120,
recommendedMin: 200,
recommendedMax: 350,
dataType: "EMAIL",
associatedText: "correo electronico"
}
}

6.8.2 Sistema de severidad

Los criterios de clasificación para los findings identificados en el proceso de análisis de usabilidad y performance de interfaces de usuario permiten priorizar acciones correctivas y orientar la mejora continua de los sistemas evaluados. Se estructuran en tres niveles de severidad: High (), Medium () y Low (), cada uno asociado a un impacto específico en la experiencia del usuario y la seguridad de los datos.

High (): Impacto crítico en usabilidad

Las incidencias clasificadas como High representan riesgos significativos tanto para la experiencia del usuario como para la integridad de la información gestionada por el sistema. En este nivel se incluyen:

- Datos sensibles sin protección (S06): La falta de mecanismos adecuados para salvaguardar información confidencial expone a los usuarios y a la organización a potenciales brechas de seguridad, con posibles consecuencias legales y reputacionales.
- Formatos críticos faltantes (S03): La ausencia de validación o estandarización en campos clave (como identificadores, correos electrónicos, etc.) puede generar errores sistémicos, pérdida de datos o imposibilidad de completar procesos críticos.
- Flujos excesivamente largos (S07): Procesos que requieren pasos innecesarios o redundantes afectan negativamente la eficiencia operativa, incrementan la tasa de abandono y reducen la satisfacción del usuario final.

Medium (): Degradación de eficiencia

Las findings de severidad Medium se asocian a deficiencias que, si bien no comprometen de manera crítica la seguridad o la usabilidad, generan una degradación perceptible en la eficiencia del uso del sistema. Se destacan los siguientes casos:

- Inconsistencias dimensionales (S02): Variaciones no justificadas en el tamaño de los componentes visuales pueden dificultar la navegación y la comprensión, afectando la coherencia visual y cognitiva de la interfaz.
- Campos fuera de rangos óptimos (S01): El uso de dimensiones inadecuadas en campos de entrada, tanto por exceso como por defecto, puede limitar la legibilidad y la facilidad de interacción.
- Vocabularios limitados mal implementados (S05): La presencia de listas de selección o catálogos incompletos o mal estructurados restringe la capacidad del usuario para seleccionar opciones adecuadas, incrementando el error y la frustración.

Low (): Oportunidades de mejora

Las observaciones clasificadas como Low corresponden a aspectos que, si bien no afectan directamente la funcionalidad esencial ni la seguridad, representan oportunidades para optimizar la experiencia del usuario y refinar detalles de la interfaz:

- Vínculos vagos ocasionales (S04): Enlaces o llamados a la acción poco claros pueden generar dudas, aunque su impacto es acotado si se presentan de forma esporádica.
- Variaciones menores de dimensiones (S01): Pequeñas diferencias en el tamaño de los elementos, aunque no críticas, pueden ser ajustadas para mejorar la uniformidad visual y la percepción de calidad.

Esta clasificación no solo facilita la priorización de acciones correctivas, sino que también se integra con las funciones de filtrado, agrupación y gestión de findings descritas en las secciones posteriores del documento. De este modo, el sistema permite a los equipos de desarrollo enfocar sus esfuerzos en los aspectos que más contribuyen a la mejora de la usabilidad, la seguridad y la eficiencia operativa de sus productos digitales.

6.8.3 Acciones sobre findings

[CAPTURA: Menú contextual de acciones sobre finding]

Acciones disponibles:

1. **Seleccionar**: Centra y enfoca nodo en canvas
2. **Ignorar**: Marca como ux-ignored- persistente
3. **Restaurar**: Revierte estado ignorado
4. **Confirmar como Input**: Para detectores de candidatos
5. **Ver Metadatos**: Información técnica detallada

6.8.4 Filtrado y agrupación

Los controles de filtrado ofrecen una gran flexibilidad para visualizar y gestionar los findings dentro del sistema. Permiten a los usuarios refinar la lista de elementos detectados aplicando distintos criterios, tales como:

- Por severidad: Es posible filtrar los findings según su nivel de severidad, clasificándolos en High (alto), Medium (medio) y Low (bajo). Esto ayuda a priorizar aquellos hallazgos que representan un mayor riesgo o impacto para el producto, enfocando la atención del equipo en los issues más críticos.

- Por tipo de detector: Los findings pueden ser segmentados de acuerdo al detector específico que los identificó (por ejemplo, S01-S07). Esta opción facilita la revisión especializada por áreas o tipos de anomalías, optimizando el trabajo de los expertos en cada aspecto.
- Por frame contenedor: El sistema permite filtrar los findings según el frame o módulo de la interfaz en el que se encuentran, lo que resulta útil para abordar problemas en contextos específicos del diseño o funcionalidad.
- Por estado: Los findings pueden mostrarse según su estado actual dentro del flujo de trabajo: Activos (pendientes de acción), Ignorados (descartados por decisión del equipo), o Todos (sin distinción de estado). Esto brinda una visión integral o focalizada, según las necesidades del análisis.

La agrupación inteligente complementa los filtros, permitiendo organizar los findings de manera más eficiente y significativa. Las opciones incluyen:

- Por tipo de smell: Agrupa los findings según el tipo de problema o "smell" detectado, facilitando la identificación de patrones recurrentes y la toma de decisiones estratégicas para su resolución.
- Por frame contenedor: Ordena los findings de acuerdo al módulo o sección de la interfaz donde se presentan, agilizando la resolución de issues localizados.
- Por severidad descendente: Organiza los findings empezando por los de mayor severidad, permitiendo que los elementos más críticos sean abordados en primer lugar.
- Por orden de aparición en canvas: Presenta los findings siguiendo el orden en que fueron detectados en el canvas, lo que puede ser útil para rastrear el flujo de análisis o la evolución de los issues durante el proceso de revisión.

El uso combinado de controles de filtrado y agrupación inteligente no solo optimiza la gestión de findings, sino que también potencia la capacidad del equipo para identificar tendencias, priorizar tareas y mantener un seguimiento detallado del estado de cada issue detectado. Estas funciones se integran perfectamente con las acciones operativas y los flujos de trabajo descritos en las secciones siguientes, contribuyendo así a una mejora continua en la calidad y eficiencia de los productos digitales.

6.9 Flujos de trabajo optimizados

6.9.1 Flujo de trabajo básico

[CAPTURA: Diagrama de flujo de trabajo recomendado]

1. **Preparación** (2 min):
    1. Wireframe con nomenclatura consistente
    2. Labels definidos y asociados a inputs
    3. Prototipo básico conectado
2. **Análisis inicial** (30 seg):
    1. Ejecutar "Análisis Completo"
    2. Revisar distribución de severidades
    3. Exportar baseline para documentación
3. **Corrección iterativa** (5-10 min):
    1. Corregir findings High priority
    2. Re-ejecutar detectores específicos
    3. Validar reducción de issues
4. **Refinamiento** (3-5 min):
    1. Abordar Medium priority
    2. Ignorar falsos positivos justificados
    3. Documentar decisiones de diseño
5. **Validación final** (1 min):
    1. Análisis completo final
    2. Export para handoff
    3. Registro de versión

6.10 Exportación y documentación

6.10.1 Formato CSV detallado

[CAPTURA: Exportación CSV con todas las columnas]

Estructura optimizada para exportación CSV

La exportación de datos utiliza una estructura de columnas detallada para asegurar la trazabilidad y el análisis eficiente de los findings relacionados con usabilidad. A continuación se detalla el formato sugerido:

Columnas principales

- id: Identificador único del nodo o elemento analizado.
- frame: Nombre del frame o sección donde se encuentra el issue.
- name: Identificación del componente o campo afectado.
- issue: Descripción breve del problema detectado.
- type: Tipo de issue (por ejemplo, SEMANTIC, VISUAL, etc.).
- severity: Gravedad asignada (low, medium, high).
- suggestion: Recomendación concreta de ajuste o mejora.
- metadata: Información complementaria relevante, en formato estructurado (por ejemplo, {"width":120}).
- timestamp: Fecha y hora del análisis o registro del finding.
- version: Versión de la herramienta o plataforma utilizada durante el análisis.

Ejemplo de fila

node_123,Registro,Campo Email,Ancho no ideal,SEMANTIC,medium,200-350px,"{width:120}",2025-09-14T10:30:00Z,1.2.0

Columnas adicionales sugeridas

- detector_version: Indica la versión específica del detector que identificó el issue.
- analysis_context: Define el alcance y la configuración utilizada en el análisis.
- correction_applied: Booleano para el seguimiento de si la sugerencia fue implementada.
- designer_notes: Campo libre donde el diseñador puede registrar anotaciones adicionales.

6.11.2 Formato Markdown enriquecido

[CAPTURA: Export Markdown con metadatos y métricas]

Template mejorado:

Reporte del Verificador de Smells - Verificador Analysis 29 de septiembre de 2025

Resumen ejecutivo

- **Total findings**: 28

- **Densidad**: 1.17 findings/input

- **Severidad alta**: 17 (críticos)

- **Tiempo de análisis**: 248ms

Métricas por detector

| Detector | Findings | Tiempo | Status |

|----------|----------|--------|---------|

| S01 | 17 | 630ms | ✅ Stable |

| S02 | 26 | 850ms | ✅ Stable |

| S03 | 7 | 212ms | ✅ Stable |

Findings detallados

🔴 Severidad Alta (43)

1. Campo Email 1

- **Tipo**: SEMANTIC

- **Detector**: S01

- **Elemento**: `node\_100`

- **Descripción**: Ancho no ideal para email

- **Sugerencia**: Aumentar ancho a 200-350px

6.11.3 Integration APIs (futuro)

Webhooks configurables:

```
EXPORT_INTEGRATIONS: {
slack: {
webhook_url: "...",
template: "summary",
trigger: "high_severity"
},
linear: {
api_key: "...",
project_id: "...",
auto_create_issues: true
}
}
```

6.12 Problemas frecuentes y soluciones

Problema	Síntomas	Causa raíz	Solución
No se detectan inputs	0 findings en página con campos	Nodos no cumplen heurísticas morfológicas	Verificar: Rectangle 24-60px altura, texto asociado
Falsos positivos en S02	Inputs no relacionados agrupados	Proximidad espacial ambigua	Ajustar MAX_DISTANCIA_VERTICAL o separar visualmente
Rendimiento lento	>2s en análisis completo	Archivo con >5000 nodos	Usar scope selection, reducir complejidad
Cache corrupto	Comportamiento inconsistente	Datos persistidos inválidos	Ejecutar "Reset to Defaults"
Sistema modular falla	Fallback a legacy continuo	Error en inicialización de módulos	Verificar logs consola, reportar a desarrollo

6.13 Extensión y desarrollo futuro

- Desarrollo de nuevos detectores
    - La evolución del sistema puede orientarse hacia la incorporación de detectores adicionales que aborden problemáticas comunes en el diseño, tales como la ausencia o ubicación incorrecta de etiquetas, la alineación inconsistente de elementos, el espaciado irregular y aspectos vinculados a la accesibilidad, como el contraste y el uso adecuado de placeholders. La selección y priorización de estos detectores debe responder a criterios de viabilidad técnica, impacto en la calidad y requerimientos de los usuarios.
- Expansión de la API de extensión
    - El sistema ofrece la posibilidad de extender su funcionalidad mediante el registro e integración de detectores personalizados. Esta capacidad permite adaptar la herramienta a contextos específicos, facilitando la incorporación de reglas propias de cada organización o dominio, y promoviendo la participación de la comunidad de usuarios avanzados o desarrolladores externos.

6.14 Consideraciones éticas y privacidad

6.14.1 Manejo de datos sensibles

El sistema adopta una serie de principios fundamentales para garantizar la protección y correcta gestión de los datos sensibles durante el proceso de análisis y operación. Estos principios aseguran que la privacidad de los usuarios se mantenga intacta y que ningún dato confidencial se vea comprometido en ningún momento.

Análisis sin extracción de contenido

El análisis realizado por el sistema se limita estrictamente a los metadatos estructurales, evitando por completo la extracción o manipulación del contenido original. Esto significa que el sistema no accede, interpreta ni almacena información textual o visual sensible, sino que se enfoca únicamente en la estructura y los patrones que definen la organización de los datos.

Detección basada en patrones

La detección de posibles incidencias o elementos relevantes se lleva a cabo mediante la identificación de patrones, utilizando keywords que han sido normalizadas. El sistema no trabaja con texto literal, lo que evita el riesgo de exponer información privada o confidencial durante el análisis.

Almacenamiento local de configuraciones

Todas las configuraciones generadas y utilizadas por el sistema se almacenan exclusivamente de manera local, a través de clientStorage. Esto garantiza que los datos sensibles no salgan del entorno del usuario y que la información se mantenga bajo control, evitando transferencias o accesos no autorizados.

Ausencia total de telemetría

El sistema está diseñado para operar sin enviar ningún tipo de dato a servidores externos, eliminando así cualquier riesgo asociado con la telemetría. Esta política asegura que la privacidad y la seguridad de los datos sensibles permanezcan protegidas en todo momento, ya que no se realiza ningún tipo de transmisión fuera del entorno local.

6.14.2 Buenas prácticas de privacidad

Para preservar la privacidad y reforzar la protección de los datos sensibles durante el uso del sistema, se recomienda adoptar las siguientes buenas prácticas:

- Revisar exports antes de compartir externamente:
  - Es fundamental analizar los archivos exportados antes de ser enviados fuera del entorno interno, con el objetivo de verificar que no contengan información confidencial o sensible que pudiera exponer a la organización o a los usuarios.
- Evitar nombres de capas con información personal:
  - Se debe evitar el uso de identificadores, nombres o cualquier dato personal en las capas o elementos del sistema, disminuyendo así el riesgo de revelar información privada por error al compartir recursos.

6.15 Adopción organizacional

6.15.1 Checklist de implementación

La adopción organizacional del sistema se estructura en tres fases principales, cada una con objetivos y actividades específicas para garantizar un proceso de integración progresivo y exitoso.

Fase 1: Piloto (Semana 1-2)

- Instalación inicial del sistema en el equipo core, conformado por 2 a 3 diseñadores clave.
- Configuración de los tipos específicos de dominio, permitiendo adaptar el funcionamiento del sistema a las necesidades del grupo piloto.
- Establecimiento de un preset organizacional que actúe como referencia para futuras implementaciones.
- Realización de una sesión de capacitación de una hora para asegurar el correcto uso de la herramienta desde el inicio.
- Análisis de 3 a 5 proyectos seleccionados como baseline, con el objetivo de obtener métricas iniciales y detectar oportunidades de mejora.

Fase 2: Expansión (Semana 3-4)

- Implementación del sistema en el equipo completo de diseño, ampliando el alcance y el impacto.
- Integración formal de la herramienta dentro del proceso de design review, facilitando su uso habitual y sistemático.
- Configuración de templates de exportación para estandarizar la salida y el intercambio de información.
- Definición y establecimiento de métricas objetivo que guíen el seguimiento del progreso y la efectividad del sistema.
- Documentación de las guidelines internas para asegurar la consistencia en el uso y la interpretación de resultados.

Fase 3: Optimización (Mes 2)

- Análisis detallado de las métricas de adopción recolectadas durante las fases previas para identificar áreas de mejora.
- Refinamiento de configuraciones en función de los aprendizajes y feedback obtenidos.
- Capacitación avanzada dirigida a usuarios expertos, potenciando el uso de funcionalidades avanzadas.
- Integración de la herramienta con otras existentes en el ecosistema de diseño de la organización.
- Implementación de un proceso de mejora continua para asegurar la evolución sostenida de la adopción y el rendimiento del sistema.

6.15.2 Métricas de adopción sugeridas

Para evaluar la adopción y el impacto del sistema, se proponen métricas tanto a nivel individual como organizacional:

Individuales:

- Número de análisis ejecutados por semana, lo que permite medir el nivel de uso y compromiso de cada usuario.
- Tiempo promedio dedicado a la corrección de los hallazgos, indicador de eficiencia y facilidad de uso.
- Porcentaje de findings corregidos frente a aquellos que son ignorados, reflejando el grado de acción sobre los problemas detectados.
- Mejora en el density score, como métrica de progreso en la calidad de los proyectos.

Organizacionales:

- Coverage: porcentaje de proyectos analizados, que muestra el alcance del sistema en la organización.
- Quality index: promedio de scores obtenidos, permitiendo evaluar la calidad general alcanzada.
- Efficiency: tiempo ahorrado en tareas de QA, reflejo del impacto en la productividad.
- Consistency: varianza entre equipos, indicando el nivel de homogeneidad en la aplicación de estándares.

6.16 Conclusiones y próximos pasos

6.16.1 Valor entregado

El plugin Simple Smells Detector ha logrado implementar de manera exitosa los detectores S01-S07, definidos en el Capítulo 4, aportando beneficios significativos tanto a nivel técnico como operativo.

Desde la perspectiva técnica, se destaca una arquitectura modular y extensible, tal como se describe en la Sección 5.5.1, que permite la incorporación futura de nuevas funcionalidades. El sistema ofrece un enfoque híbrido con mecanismos de fallback robustos, asegurando la continuidad del análisis incluso ante posibles fallos. Se optimizó el rendimiento, alcanzando tiempos de análisis inferiores a 500 milisegundos para 1000 nodos, y se mejoró la precisión gracias a una configuración adaptativa que permite ajustar los parámetros a las necesidades del proyecto.

En el plano operacional, el plugin se integra de forma natural en el flujo de trabajo de Figma, facilitando ciclos de feedback inferiores a 10 minutos. Esto contribuye a una reducción promedio superior al 80% en el density score, reflejando mejoras sustanciales en la calidad de los prototipos. Además, se automatizó la documentación de los quality gates, simplificando el seguimiento de los estándares de calidad durante el proceso de diseño.

6.16.2 Limitaciones reconocidas

A pesar de los avances logrados, existen limitaciones que es importante reconocer. En el aspecto técnico, el sistema depende de una nomenclatura consistente para el funcionamiento de los detectores semánticos, lo que puede restringir su aplicabilidad en proyectos con convenciones diversas. La cobertura actual se limita a nodos estructurados, específicamente Rectangle, Frame y Text, y el análisis realizado es estático, sin validar la interacción real del usuario.

Desde el punto de vista operacional, la configuración avanzada del plugin puede presentar una curva de aprendizaje para los usuarios, requiriendo disciplina en los procesos de corrección iterativa. Además, el alcance de la herramienta se circunscribe a aspectos de usabilidad estructural, sin abordar de manera integral temas de accesibilidad.

6.16.3 Evolución continua

La evolución del sistema contempla líneas claras de desarrollo técnico y de adopción organizacional. En el roadmap técnico para el cuarto trimestre de 2025 se proyecta la incorporación de nuevos detectores, la integración de técnicas de machine learning para una clasificación semántica más avanzada, el desarrollo de una API que facilite integraciones empresariales y el soporte para design systems complejos.

En cuanto a la adopción organizacional, se planea implementar un programa de entrenamiento estructurado, definir métricas de retorno de inversión en calidad de diseño, integrar el plugin con herramientas de project management y fomentar una comunidad de práctica para el intercambio de configuraciones y experiencias.

De esta manera, el Capítulo 6 sienta las bases operacionales para una adopción sostenible del sistema, cumpliendo con el objetivo OE7 de generar directrices prácticas que vinculan los hallazgos técnicos con flujos de trabajo ágiles y efectivos. La combinación de ejemplos prácticos, métricas cuantificables y procesos estructurados facilita la transición de la investigación académica hacia una herramienta aplicada en entornos profesionales de producción.

[CAPTURA: Dashboard final mostrando métricas de éxito de implementación]

Fin del Capítulo 6 - Instalación, Configuración y Uso del Plugin Simple Smells Detector

Capítulo 7. Trabajo Futuro y Cierre Teórico

7.1 Objetivo del capítulo

Identificar las principales limitaciones del enfoque actual y delinear líneas de evolución técnica, investigativa y práctica que amplíen la cobertura y robustez del sistema de detección automática de problemas de usabilidad en prototipos de diseño de interfaces digitales.

7.2 Síntesis de aportes previos

- Cap. 1–2: Fundamentos conceptuales y motivación (brecha entre evaluación temprana y herramientas automatizadas ligeras).
- Cap. 3: Justificación comparativa de Figma como plataforma óptima.
- Cap. 4: Taxonomía ampliada con clasificación de viabilidad (S01–S25).
- Cap. 5: Arquitectura implementada y propuesta de modularización estandarizada.
- Cap. 6: Guía operativa para adopción y uso sostenible.

7.3 Limitaciones actuales

Área de Enfoque	Limitación Específica	Impacto Observado	Causa Raíz o Motivo Principal
Cobertura	No se incluye el análisis de accesibilidad visual avanzada.	Resulta en un menor alcance inclusivo del análisis general.	Falta un análisis detallado del contraste cromático y la evaluación de roles semánticos (ARIA).
Análisis Semántico	Existe una dependencia estricta de etiquetas textuales explícitas.	Produce Falsos Positivos (FP) o Falsos Negativos (FN), especialmente en el análisis de wireframes abstractos.	Ausencia de un mecanismo de meta-anotaciones semánticas o datos estructurados.
Validación Empírica	No se dispone de un dataset etiquetado público para pruebas.	Las métricas de rendimiento y exactitud no son reproducibles externamente.	El esfuerzo manual de catalogación y etiquetado del dataset aún está pendiente.
Evolución y Trazabilidad	Los hallazgos y reportes no tienen un versionado formal.	Dificulta replicar y comparar los resultados obtenidos en diferentes momentos históricos.	Falta un campo version uniforme y obligatorio en la estructura de los reportes.
Internacionalización	El léxico y vocabulario están limitados a español (ES) e inglés (EN) básicos.	Genera un riesgo de Falsos Negativos (FN) en el análisis de dominios sectoriales o técnicos específicos.	Ausencia de diccionarios sectoriales o glosarios terminológicos especializados.
Análisis Dinámico	No se realiza un análisis de flujos de interacción o comportamiento temporal.	Los problemas de retroalimentación o errores basados en la secuencia de acciones no son detectables.	La herramienta se basa en prototipos estáticos que carecen de la definición de eventos y transiciones.

Líneas de trabajo futuro (técnicas)

Las futuras líneas de investigación y desarrollo técnico se orientan a abordar las limitaciones identificadas en los apartados previos, especialmente aquellas vinculadas a la replicabilidad de resultados, la internacionalización y el análisis dinámico de los sistemas evaluados.

- Normalización y versionado de reportes: Desarrollar un esquema unificado y obligatorio para la versión de los reportes, facilitando así la comparación de resultados a través del tiempo y entre diferentes implementaciones.
- Internacionalización y especialización terminológica: Incorporar diccionarios sectoriales y glosarios técnicos multilingües, permitiendo un análisis más preciso en dominios específicos y reduciendo el riesgo de falsos negativos derivados de vocabulario limitado o ambiguo.
- Análisis dinámico de interacción: Implementar mecanismos para el seguimiento de flujos de usuario y la detección de errores basados en la secuencia de acciones, mediante la definición de eventos y transiciones en prototipos interactivos.
- Evaluación automatizada de retroalimentación: Desarrollar herramientas que permitan la marcación manual y el registro persistente de falsos positivos y negativos, facilitando el aprendizaje incremental del sistema y la reducción de sesgos.
- Optimización del rendimiento: Introducir pipelines de análisis diferencial y procesamiento incremental que permitan escalar el análisis a grandes volúmenes de datos sin sacrificar precisión ni tiempos de respuesta.
- Paralelización y agregación de resultados: Favorecer la ejecución concurrente de detectores independientes, con agregación inteligente de resultados, para mejorar la eficiencia y la cobertura de los análisis.

Estas líneas de trabajo buscan fortalecer la robustez, la adaptabilidad y la precisión de las herramientas desarrolladas, alineándose con los desafíos detectados en la replicabilidad histórica, la cobertura terminológica y el análisis de procesos dinámicos.

7.5 Extensiones basadas en datos

Las extensiones propuestas a continuación se centran en aprovechar los datos generados durante el proceso de evaluación para mejorar la eficiencia, precisión y adaptabilidad de las herramientas. Cada propuesta incluye una descripción, requisitos adicionales para su implementación y posibles riesgos asociados.

Retroalimentación supervisada

Se plantea la incorporación de un mecanismo mediante el cual los diseñadores puedan marcar manualmente los falsos positivos (FP) y falsos negativos (FN) detectados en el sistema. Este proceso requiere la persistencia del juicio experto asociado a cada hallazgo, permitiendo así un aprendizaje incremental y guiado por experiencia real. Sin embargo, este enfoque puede verse afectado por el sesgo individual de los etiquetadores, lo que podría influir en la calidad de las anotaciones.

Calibración adaptativa

La calibración adaptativa consiste en ajustar los umbrales de detección de acuerdo con la distribución observada de los datos en cada proyecto. Para implementar esta estrategia, resulta necesario mantener un historial anónimo de métricas asociadas a los proyectos, permitiendo adaptar la herramienta a contextos específicos. El principal riesgo de esta propuesta reside en la posibilidad de sobreajustar los parámetros a patrones particulares (overfitting), lo que podría limitar la generalización de los resultados.

Perfilado de proyectos

Esta extensión propone la generación de estadísticas agregadas, tales como el promedio de componentes y la densidad de elementos, a partir de los análisis realizados. Para ello, es imprescindible almacenar un resumen (snapshot) de las métricas obtenidas en cada proyecto. Un riesgo potencial asociado a este enfoque es la posible afectación de la privacidad si estos datos se exportan fuera del equipo de trabajo.

Análisis predictivo

El análisis predictivo tiene como objetivo detectar tendencias en la evolución de los prototipos a lo largo del tiempo. Para lograrlo, se requiere la integración de la herramienta con un sistema de versionado que permita rastrear los cambios y analizar su impacto. Sin embargo, este tipo de análisis implica una mayor complejidad tanto en la implementación como en el mantenimiento de la solución.

7.6 Agenda Integrada de Investigación, Validación y Desarrollo del Sistema de Inspección Automática en UX

Lo siguiente se configura como una continuidad lógica y argumental de las líneas expuestas en los capítulos anteriores, donde se fundamentó la necesidad de mecanismos automáticos para la detección temprana de problemas de usabilidad en prototipos de interfaces. El abordaje propuesto busca no sólo profundizar los procesos de validación empírica, sino también establecer marcos de referencia robustos para la integración, evaluación y evolución del sistema, contemplando tanto los desafíos técnicos como los requerimientos éticos y normativos propios de la disciplina UX.

7.6.1 Preguntas de Investigación Prioritarias

Se plantea un conjunto de interrogantes orientadores cuyo abordaje resulta central para el avance del campo y la consolidación metodológica de la solución propuesta:

- ¿Cuál es la proporción real de incidencias de usabilidad identificables exclusivamente mediante el análisis estático de la estructura visual y textual, y cómo varía esta capacidad según el dominio de aplicación?
- ¿Existe un umbral crítico de densidad visual por encima del cual la eficacia de las intervenciones correctivas decae significativamente, y cómo se modela la curva de saturación resultante?
- ¿En qué medida la acumulación simultánea de hallazgos (“smells”) genera fatiga de alerta en los equipos de diseño, y cuáles son los parámetros óptimos para maximizar la adopción y el aprovechamiento efectivo de la herramienta?
- ¿Qué niveles mínimos de precisión y exhaustividad (precision, recall) son requeridos para que la integración del sistema en los procesos de Definition of Done resulte aceptada y sostenible por equipos interdisciplinarios?

7.6.2 Protocolo de Validación Empírica

La validación experimental del sistema se articula en fases secuenciales, orientadas a garantizar la robustez y generalización de los resultados:

- Selección de un conjunto piloto compuesto por proyectos reales, representativos de diversas tipologías y contextos de uso, con un volumen total de entradas anotadas que asegure la significatividad estadística.
- Establecimiento de una línea base de desempeño para los detectores implementados (S01–S07), midiendo sistemáticamente precision y recall bajo condiciones controladas.

- Iteración sobre los umbrales de detección mediante ciclos sucesivos de ajuste, con el objetivo de estabilizar métricas y reducir la variabilidad interproyecto.
- Incorporación progresiva de nuevos detectores (por ejemplo, S08–S10, S12, S14, S20) y reevaluación integral del sistema, documentando los efectos de cada ampliación.
- Publicación de un esquema de dataset anonimizado, carente de información sensible, para fomentar la replicabilidad y el escrutinio académico externo.

7.6.3 Implicancias para la Práctica Profesional en UX

La integración de la evaluación heurística automatizada dentro del ciclo de diseño ágil redefine la dinámica profesional al permitir la detección y corrección de deficiencias sin depender exclusivamente de etapas posteriores de testeo con usuarios. Este enfoque favorece la estandarización de los debates de diseño, ya que los hallazgos exportados constituyen artefactos objetivos y auditables. Además, facilita el proceso de onboarding de nuevos integrantes al exponer explícitamente las convenciones y patrones promovidos, contribuyendo a la homogeneización y trazabilidad de criterios dentro de los equipos.

7.6.4 Consideraciones Éticas y de Privacidad

La gestión responsable de datos constituye un pilar fundamental en el desarrollo del sistema. Por ello, se establecen políticas estrictas que prohíben la captura de contenido textual completo en campos sensibles, privilegiando el almacenamiento de etiquetas normalizadas. Se habilitan mecanismos de anonimización de exportaciones, tales como el reemplazo de identificadores visuales por hashes, para asegurar la protección de la identidad y la privacidad de los usuarios. Asimismo, se promueve la transparencia informativa mediante la indicación explícita de la versión de heurísticas y las limitaciones inherentes a cada exportación, previniendo interpretaciones erróneas o extrapolaciones fuera del alcance declarado.

7.6.5 Alineación con Estándares Internacionales

La convergencia con marcos internacionales de referencia se plantea como un objetivo estratégico. En este sentido, se prevé la integración progresiva de requisitos derivados de las pautas WCAG 2.x (especialmente en contraste y legibilidad), la alineación de los detectores con principios de ergonomía definidos por la norma ISO 9241 (coherencia, carga cognitiva), y la clasificación sistemática de hallazgos conforme a las heurísticas de Nielsen (visibilidad, prevención de errores). Cada una de estas acciones será acompañada por la documentación de los criterios de mapeo y la evaluación de su impacto en la práctica.

7.6.6 Evolución Arquitectónica Propuesta

La evolución de la arquitectura del sistema se concibe como un proceso iterativo, donde cada etapa incorpora mejoras dirigidas a la escalabilidad, portabilidad y eficiencia operativa:

- Incorporación de una capa de adaptación (adapter) que facilite la portabilidad y el aislamiento respecto de las dependencias de la API.
- Implementación de un registro centralizado de detectores y métricas, ampliando las capacidades de observabilidad y permitiendo la activación selectiva según contexto.
- Despliegue de un mecanismo de caché de geometría, orientado a la reducción drástica de los tiempos de análisis en escenarios de alta complejidad.
- Adopción de un motor de reglas declarativas (por ejemplo, en formato JSON), posibilitando la incorporación ágil de nuevos smells sin requerir modificaciones en el código base.
- Desarrollo de un modo experimental tipo sandbox, que habilite la experimentación controlada (A/B testing) de heurísticas antes de su promoción a versiones estables.

7.6.7 Métricas de Madurez del Sistema

La progresión hacia la madurez se evalúa mediante una escala multinivel, asociada a indicadores verificables:

- Nivel 1 – Inicial: Disponibilidad de detectores manuales básicos y capacidad de análisis en menos de 1 segundo por cada 200 nodos procesados.
- Nivel 2 – Estandarizado: Generación de hallazgos estructurados con presets reutilizables, exportación reproducible y versionado de heurísticas.
- Nivel 3 – Medible: Incorporación de métricas internas de precisión y exhaustividad, con informes detallados por release.
- Nivel 4 – Adaptativo: Ajuste dinámico de umbrales y retroalimentación continua, evidenciado por una reducción sostenida de falsos positivos.
- Nivel 5 – Optimizado: Integración de modelos predictivos ligeros y estabilización de la curva de mejora marginal, garantizando eficiencia y robustez operativa.

7.6.8 Riesgos Emergentes y Estrategias de Mitigación

El despliegue y evolución del sistema conllevan la aparición de riesgos específicos, cuya anticipación y gestión resultan imprescindibles:

- Deriva semántica: La emergencia de nuevos patrones de UI puede erosionar la efectividad de los detectores; se propone una revisión sistemática y trimestral de los diccionarios de referencia.
- Incremento de ruido: La incorporación indiscriminada de detectores no calibrados puede aumentar los falsos positivos; se sugiere una estrategia de lanzamiento escalonado y etiquetado explícito de funcionalidades en estado beta.
- Fragmentación de versión: La proliferación de forks internos divergentes amenaza la coherencia evolutiva; la publicación de una especificación mínima y de una suite de tests oficiales servirá como mecanismo de alineación.
- Percepción de sustitución: El riesgo de que los equipos interpreten la herramienta como reemplazo de las pruebas con usuarios requiere una documentación explícita del alcance y limitaciones del sistema.

7.14 Síntesis conclusiva

El análisis realizado a lo largo del capítulo destaca el valor de incorporar una capa de evaluación automática centrada en señales estructurales y textuales detectadas en etapas tempranas del diseño. Esta aproximación permite reducir significativamente los costos asociados al retrabajo y fomenta una mayor disciplina en el desarrollo de formularios y flujos de interacción. Sin embargo, el potencial de expansión de esta capa —abarcando aspectos como accesibilidad, semántica enriquecida, densidad informativa y jerarquía tipográfica— depende de la disponibilidad de datos relevantes y de la capacidad para gestionar y controlar el ruido, especialmente en lo que respecta a los falsos positivos (FP). Por ello, se define una ruta de evolución escalonada que privilegia la robustez y la transparencia del sistema por encima de la incorporación prematura de mayor complejidad.

7.15 Conclusiones del capítulo

El presente capítulo consolida el espacio de mejora futura, alineando las propuestas con los objetivos OE1–OE6. Se presenta un plan evolutivo, tanto en el aspecto técnico como metodológico, que equilibra cuidadosamente el valor incremental y los riesgos asociados. Esta planificación permite avanzar en la investigación y el desarrollo sin comprometer la estabilidad alcanzada. Además, el capítulo actúa como un puente directo hacia las conclusiones generales y los anexos, donde se ofrece la trazabilidad completa del trabajo, así como referencias y materiales complementarios que respaldan y enriquecen el proceso de implementación.

Capítulo 8. Conclusiones y Trabajos Futuros

8.1. Síntesis de la Contribución y Respuesta a las Preguntas de Investigación

La presente tesina ha abordado una brecha crítica en el ciclo de vida del desarrollo de software: la ausencia de un mecanismo automatizado, integrado en el entorno de prototipado, para la detección temprana de problemas de usabilidad. El problema central, como se delineó en la introducción, radica en que la identificación tardía de estos problemas, denominados *usability smells*, conduce a la acumulación de deuda de experiencia de usuario (UX) y a sobrecostos significativos durante las fases de implementación y mantenimiento.[1] La investigación empírica ha demostrado consistentemente que la mayoría de las evaluaciones de usabilidad se concentran en la fase de implementación, la más costosa para introducir cambios, con aproximadamente el 90% de los estudios aplicando evaluaciones en esta etapa tardía.[1] Este paradigma reactivo contrasta con los principios de desarrollo ágil y diseño centrado en el usuario, que abogan por una retroalimentación temprana y continua.

Para dar respuesta a esta problemática, la contribución principal de este trabajo es el diseño, implementación y documentación de un artefacto software: un plugin para la plataforma de diseño colaborativo Figma. Este plugin materializa un cambio de paradigma, desplazando el análisis de usabilidad desde las fases posteriores al despliegue —que dependen del análisis de logs de interacción de usuarios reales [1]— hacia la fase de diseño conceptual y prototipado de baja y media fidelidad. A continuación, se sintetiza cómo el desarrollo de esta herramienta ha permitido responder a las preguntas de investigación que guiaron el proyecto.

En respuesta a la **Pregunta de Investigación 1 (PI1)**: *¿Cuáles usability smells presentan señales estructurales observables (como tamaño, proximidad, nomenclatura o texto) suficientes para permitir su detección automática en prototipos que no cuentan con lógica ejecutable?*, este trabajo ha demostrado que un subconjunto relevante de *usability smells* posee, efectivamente, indicadores estructurales y cuantificables. Se ha construido una taxonomía de *smells* automatizables, incluyendo categorías como *Unformatted Input*, *Free Input for Limited Values* y *Short Input*, cuya detección se basa en el análisis de propiedades geométricas, contenido textual y jerarquía de capas del prototipo. La implementación de detectores heurísticos para estos casos valida la hipótesis de que la analogía conceptual entre los *code smells* y los *usability smells* no es meramente teórica, sino que es prácticamente operacionalizable en artefactos estáticos, sin necesidad de código ejecutable o interacción dinámica. Este hallazgo solidifica la base teórica para una nueva clase de herramientas de análisis en la fase de diseño, demostrando que el "olor" a un problema de usabilidad es una propiedad inherente al artefacto de diseño en sí mismo, y no solo una propiedad emergente de su uso.

Respecto a la **Pregunta de Investigación 2 (PI2)**: *¿Qué plataforma de prototipado proporciona las mejores condiciones (API, modelo de objetos, adopción, extensibilidad) para implementar detectores no intrusivos?*, el análisis comparativo de herramientas (Capítulo 3) justificó la elección de Figma. La decisión se fundamentó en la superioridad de su API de plugins, que ofrece un acceso granular y no intrusivo al árbol de nodos del diseño, permitiendo la inspección de propiedades geométricas, textuales y de prototipado. Adicionalmente, su naturaleza

colaborativa y su amplia adopción en la industria la posicionan como un ecosistema ideal para la diseminación y el impacto de una herramienta de esta naturaleza, cumpliendo con los requisitos para una implementación robusta y escalable.

En cuanto a la **Pregunta de Investigación 3 (PI3)**: *¿Cuál es el conjunto mínimo viable de heurísticas que logra equilibrar una adecuada cobertura con una baja tasa de falsos positivos en entornos de diseño ágiles?*, la investigación abordó este desafío mediante un proceso de diseño iterativo y calibración. Se definió un conjunto inicial de detectores para los *smells* más prevalentes y con señales más claras, como inconsistencias en formularios y uso de texto libre para valores restringidos. El equilibrio entre la cobertura (sensibilidad) y la tasa de falsos positivos (precisión) se gestionó mediante la parametrización de umbrales y la contextualización de las reglas. Se reconoce que este equilibrio es uno de los desafíos centrales en cualquier método de evaluación automatizada , y el conjunto de heurísticas implementado representa una solución pragmática y validada conceptualmente, diseñada para ofrecer valor inmediato en flujos de trabajo ágiles sin abrumar al diseñador con alertas irrelevantes.

Finalmente, para la **Pregunta de Investigación 4 (PI4)**: *¿De qué manera es posible modelar y documentar los detectores para facilitar su extensión incremental, incorporando nuevos tipos de datos, reglas o métricas?*, el plugin se diseñó con una arquitectura modular y extensible. El motor de heurísticas se desacopló de la interfaz de usuario y del núcleo de análisis, y se implementó un modelo de detectores basado en patrones que facilita la incorporación de nuevas reglas. La documentación asociada provee una guía clara para que la comunidad pueda contribuir con nuevos detectores, asegurando la sostenibilidad y evolución del artefacto más allá del alcance de esta tesina y cumpliendo con uno de los objetivos específicos del proyecto.

8.2. Discusión de los Aportes en el Contexto de la Ingeniería de Usabilidad

La contribución de esta tesina trasciende la mera creación de un artefacto software. Al situar el plugin en el contexto más amplio de la ingeniería de usabilidad y el desarrollo de software, emergen aportes significativos que dialogan con los desafíos teóricos y prácticos de la disciplina.

8.2.1. La Detección Temprana como Estrategia de Mitigación de Deuda de UX

El principal aporte conceptual de este trabajo es la materialización de una herramienta para la gestión proactiva de la deuda de UX. La literatura define este concepto como el conjunto de decisiones de diseño subóptimas que, tomadas para acelerar la entrega, generan costos acumulativos a largo plazo en forma de retrabajo, insatisfacción del usuario y deterioro de la calidad percibida (Baltes, 2021; DaFonseca, 2019). El plugin aborda la raíz de este problema al intervenir en la fase donde se originan muchas de estas decisiones: el prototipado.

Al identificar *usability smells* antes de que se "cristalicen" en código, la herramienta previene la propagación de la deuda al ciclo de desarrollo. Este enfoque proactivo contrasta marcadamente con los métodos de detección basados en análisis de logs de aplicaciones en producción, que son inherentemente reactivos. Como se ilustra en la Tabla 8.1, el desplazamiento del análisis hacia la izquierda ("shift-left") en el ciclo de vida no solo reduce drásticamente el costo de corrección, sino que también transforma la naturaleza del feedback, pasando de una corrección de problemas existentes a una prevención de su introducción. Este posicionamiento alinea la gestión de la calidad de la UX con las mejores prácticas de la ingeniería de software moderna, donde la prevención de defectos es siempre preferible a su detección y corrección tardía.

Tabla 8.1: Evolución de los Enfoques de Detección Automatizada de Usability Smells

Característica	**Enfoques Basados en Logs (e.g., USF, MUSE)**	**Enfoque Basado en Prototipos (Esta Tesina)**
Etapa del Ciclo de Vida	Post-despliegue / Producción	Pre-implementación / Diseño
Fuente de Datos	Logs de interacción de usuarios reales (eventos UI)	Prototipos estáticos (geometría, texto, metadatos de capas)
Tipo de Análisis	Dinámico / Comportamental	Estático / Estructural
Smells Detectables (Ej.)	*Missing Feedback*, *Required Inefficient Actions*, *Abandoned Form*	*Unformatted Input*, *Short Input*, *Inconsistencias de Formulario*
Naturaleza del Feedback	Reactivo (corrige problemas existentes)	Proactivo (previene la introducción de problemas)
Dependencia	Requiere una aplicación funcional y tráfico de usuarios	Requiere un prototipo bien estructurado en Figma
Coste de Corrección	Alto (requiere modificar código en producción)	Bajo (requiere modificar el diseño en el prototipo)

8.2.2. Integración en Flujos de Trabajo Ágiles y de Diseño Centrado en el Usuario (DCU)

La literatura ha documentado extensamente la fricción existente entre la velocidad de los ciclos de desarrollo ágil y la naturaleza a menudo más pausada y reflexiva de las prácticas de Diseño Centrado en el Usuario (DCU). Las evaluaciones de usabilidad tradicionales, como las

pruebas con usuarios, aunque invaluables, son costosas en tiempo y recursos, y pueden convertirse en un cuello de botella en sprints de dos semanas.

El plugin propuesto actúa como un "artefacto puente", facilitando la integración de ambas culturas. Proporciona una forma de "inspección de usabilidad continua" que opera en paralelo al sprint de diseño, ofreciendo retroalimentación inmediata y de bajo costo directamente en la herramienta del diseñador. Esto empodera a los equipos de diseño para iterar sobre problemas de usabilidad con la misma agilidad con la que iteran sobre aspectos visuales o funcionales. En lugar de esperar un informe de un experto en usabilidad al final de un ciclo, los diseñadores reciben alertas en tiempo real, lo que les permite tomar decisiones informadas sin interrumpir su flujo creativo. De esta manera, la herramienta no solo acelera el ciclo de feedback, sino que también democratiza una parte del conocimiento de usabilidad, integrándolo como una capa de calidad automatizada dentro del proceso de diseño.

8.2.3. Del "Smell" a la Refactorización: Catalizando el Ciclo de Mejora Continua

Aunque el plugin es una herramienta de diagnóstico, su valor fundamental reside en su capacidad para catalizar el ciclo de mejora continua. La identificación de un *usability smell* es el detonante necesario para una acción de refactorización de UX, definida como una transformación en la interfaz que mejora su calidad de uso sin alterar la funcionalidad subyacente. Al señalar de forma automática y objetiva un *smell* como *Free Input for Limited Values* , el plugin proporciona una justificación clara y basada en evidencia para que el diseñador considere aplicar una refactorización específica, como *Turn Input into Select* o *Turn Input into Radios*.

Esto convierte al plugin en el primer paso crítico dentro de un ciclo de vida de mejora de la usabilidad más amplio, como el propuesto por enfoques que combinan pruebas A/B y refactorización o herramientas que exploran soluciones alternativas directamente en el navegador. El artefacto de esta tesina, por lo tanto, no es un fin en sí mismo, sino un habilitador que alimenta con datos objetivos el proceso de toma de decisiones de diseño y refactorización.

Este enfoque representa un cambio fundamental en el locus de control del aseguramiento de la calidad de la usabilidad. Tradicionalmente, la evaluación ha sido una actividad delegada a expertos en usabilidad (evaluación heurística) o a procesos de investigación de usuarios (pruebas de usabilidad) que ocurren fuera del entorno de diseño principal. Herramientas de análisis de logs, aunque automatizadas, todavía requieren que un analista interprete los resultados. El plugin de Figma, en cambio, funciona de manera análoga a un corrector ortográfico o un *linter* de código para diseñadores. Democratiza un nivel básico de conocimiento de usabilidad, embebiéndolo directamente en la herramienta de creación. Esto tiene implicaciones profundas: puede alterar la "Definición de Terminado" (*Definition of Done*) para las tareas de diseño, fomentar una mayor conciencia sobre la usabilidad entre los diseñadores y, crucialmente, liberar a los escasos expertos en usabilidad para que se concentren en desafíos de UX más complejos, estratégicos y semánticos que escapan a la automatización.

8.3. Limitaciones del Estudio

Un análisis riguroso de la contribución de esta investigación exige un reconocimiento transparente de sus limitaciones. Estas fronteras, delineadas en el alcance del proyecto, no disminuyen el valor del trabajo realizado, sino que lo contextualizan y abren caminos para futuras investigaciones.

8.3.1. Limitaciones de Alcance y Enfoque Metodológico

La principal limitación del plugin reside en su enfoque deliberado en *smells* estáticos y estructurales. Si bien esta especialización es lo que le permite intervenir de manera temprana y eficaz, también implica que es ciego a toda una clase de problemas de usabilidad de naturaleza dinámica y comportamental. Cuestiones que emergen de la interacción real del usuario, como la falta de retroalimentación a una acción (*Missing Feedback*), la presencia de acciones ineficientes requeridas para completar una tarea (*Required Inefficient Actions*) o el abandono de formularios complejos (*Abandoned Form*), están fundamentalmente fuera de su alcance, ya que requieren el análisis de secuencias de eventos en una aplicación funcional.

Asimismo, la validación de la herramienta se ha centrado en una revisión experta y en la coherencia conceptual con la literatura existente, en lugar de una validación empírica a gran escala con usuarios finales. No se han realizado estudios cuantitativos para medir el impacto del plugin en métricas de producto, como la reducción del tiempo de desarrollo o la mejora en las tasas de conversión. Por lo tanto, la efectividad del plugin se establece en términos de su capacidad para detectar problemas predefinidos en prototipos, no en su impacto medido en el producto final o en la experiencia del usuario.

8.3.2. Dependencia de la Plataforma y de las Convenciones de Diseño

La eficacia del plugin está intrínsecamente ligada al ecosistema de Figma y a la calidad del prototipo que se analiza. Las limitaciones técnicas de la API de Figma pueden restringir la capacidad de detectar ciertos *smells* más avanzados, particularmente en el ámbito de la accesibilidad, como el contraste de color o la correcta implementación de atributos ARIA, que no siempre son accesibles desde el modelo de datos del plugin.

Más importante aún, la precisión de las heurísticas depende de que los diseñadores sigan convenciones razonables de nomenclatura de capas y estructuración de sus archivos. Un prototipo con una jerarquía de capas desorganizada o con nombres de componentes genéricos (e.g., "Rectangle 128") inevitablemente reducirá la efectividad de los detectores, pudiendo generar tanto falsos negativos (problemas no

detectados) como falsos positivos. La herramienta no puede inferir la intención del diseñador más allá de la evidencia estructural que este proporciona.

8.3.3. Validez de las Heurísticas y el Equilibrio entre Precisión y Cobertura

La definición de heurísticas y sus umbrales correspondientes (e.g., el número máximo de campos en un formulario antes de considerarlo "complejo") es un proceso que implica un compromiso subjetivo. No existen umbrales universalmente aceptados para muchos de estos *smells*, de la misma manera que no hay un consenso absoluto sobre el "tiempo de espera tolerable" para la carga de una página web o la cantidad de capas de navegación que constituyen el *smell Too Many Layers*. La implementación actual representa un equilibrio calibrado, pero esta calibración puede requerir ajustes para diferentes contextos de proyecto, dominios de aplicación o guías de estilo específicas de una organización.

Estas limitaciones subrayan una tensión fundamental en el campo de la evaluación automatizada de la usabilidad: el compromiso entre un análisis *temprano, escalable y superficial* (el enfoque de esta tesina) y un análisis *tardío, intensivo en recursos y profundo* (el enfoque de las pruebas con usuarios y el análisis de logs). El valor único del plugin reside en su capacidad para identificar problemas comunes y estructurales de forma muy temprana y con un costo marginal cercano a cero. Sin embargo, su naturaleza estática le impide capturar los aspectos contextuales, basados en tareas y emocionales de la UX, que métodos como las pruebas con usuarios están diseñados para descubrir. Por lo tanto, el plugin no debe ser visto como un reemplazo de otros métodos de evaluación, sino como un complemento crucial. Su función es la de depurar los "errores sintácticos" de la usabilidad, permitiendo que los expertos humanos concentren su valioso tiempo en los desafíos "semánticos" y "pragmáticos" del diseño de la experiencia de usuario.

8.4. Líneas de Trabajo Futuro e Investigación

El desarrollo del plugin presentado en esta tesina establece una base sólida sobre la cual se pueden construir futuras líneas de investigación y desarrollo. Las limitaciones identificadas no son callejones sin salida, sino invitaciones a expandir y profundizar el alcance y la inteligencia de la herramienta.

8.4.1. Expansión del Catálogo de Detectores y Heurísticas

Una dirección natural para la evolución del plugin es la expansión de su catálogo de detectores para abarcar *smells* más complejos y de diferentes dominios. Un área prioritaria es la accesibilidad. Inspirándose en trabajos sobre refactorización para la accesibilidad [1], se podrían desarrollar heurísticas para detectar problemas comunes como el uso de texto de marcador de posición (*placeholder*) en lugar de etiquetas persistentes, la ausencia de etiquetas asociadas a campos de formulario o, si la API de Figma lo permite en el futuro, la verificación de ratios de contraste de color básicos. Otra vertiente de expansión podría centrarse en principios de diseño visual, como la consistencia en el espaciado y la alineación de elementos (consistencia interna) o la desviación respecto a un sistema de diseño predefinido (consistencia externa).

8.4.2. Hacia la Sugerencia de Refactorizaciones y la Cuantificación del Impacto

El siguiente paso lógico en la evolución del plugin es pasar de una herramienta puramente diagnóstica a una prescriptiva. Al detectar un *smell*, el sistema no solo debería señalar el problema, sino también sugerir una o más refactorizaciones de UX como posibles soluciones. Por ejemplo, ante un *smell* de *Free Input for Limited Values*, el plugin podría proponer las refactorizaciones *Turn Input into Select* o *Turn Input into Radios*.

Para ayudar al diseñador a elegir entre múltiples soluciones, se podría integrar un modelo predictivo basado en métricas como el *Interaction Effort*. Este modelo podría ofrecer una estimación cuantitativa de la reducción del esfuerzo de interacción que cada refactorización sugerida podría aportar. De esta manera, el diseñador recibiría no solo una alerta, sino una recomendación accionable y respaldada por una predicción de su impacto potencial, enriqueciendo significativamente el proceso de toma de decisiones.

8.4.3. Validación Empírica y Estudios de Campo

Para establecer rigurosamente el impacto del plugin en la práctica profesional, es imperativo llevar a cabo estudios de validación empírica. Se propone el diseño de un experimento controlado, similar a una prueba A/B, donde un grupo de diseñadores utilice el plugin para realizar una tarea de diseño, mientras que un grupo de control la realiza sin él. Las métricas a evaluar incluirían la cantidad de problemas de usabilidad identificados y corregidos en la fase de prototipado, una medición de la deuda de UX residual en el producto final (por ejemplo, mediante una evaluación heurística posterior), y datos cualitativos sobre la percepción de la utilidad y la integración de la herramienta en el flujo de trabajo de los diseñadores. Este tipo de estudio proporcionaría evidencia cuantitativa y cualitativa robusta sobre el valor real del enfoque de detección temprana.

8.4.4. Integración de Inteligencia Artificial para Detección Semántica

Para superar las limitaciones de las heurísticas puramente estructurales, como se anticipó en la introducción de esta tesina, la integración de técnicas de inteligencia artificial (IA) representa la frontera más prometedora. Se podría explorar el uso de modelos de Procesamiento del

Lenguaje Natural (PLN) para analizar la claridad semántica y la consistencia de las etiquetas y los textos de instrucción. Esto permitiría detectar *smells* como *Misleading Link* no solo por su estructura, sino por el uso de textos ambiguos como "Haga clic aquí". Del mismo modo, modelos de visión por computadora podrían ser entrenados para identificar problemas de desorden visual (*visual clutter*) o inconsistencias estilísticas que se desvían de un sistema de diseño aprendido. La IA permitiría al plugin abordar una clase de *smells* semánticos y contextuales que actualmente están fuera de su alcance.

La trayectoria futura de este trabajo apunta hacia la creación de un "Asistente de Diseño Inteligente", un sistema que no solo señala errores, sino que colabora activamente con el diseñador. El plugin actual es un sistema basado en reglas. Las líneas futuras propuestas introducen sugerencias prescriptivas (refactorizaciones), análisis predictivo (esfuerzo de interacción) y reconocimiento avanzado de patrones (IA). La combinación de estos elementos podría dar lugar a una herramienta capaz de dialogar con el diseñador: "He detectado un *smell* de 'Entrada Libre' en el campo 'País'. La refactorización 'Convertir en Selector' probablemente reduciría el esfuerzo de interacción en un 30%. ¿Desea que genere una variante del componente con este cambio?". Esto transformaría la herramienta de un crítico pasivo a un socio activo en el proceso de diseño, representando una evolución significativa en el paradigma del diseño asistido por computadora para la usabilidad.

8.5. Conclusión Final

Esta tesina ha abordado con éxito el diseño, la implementación y la fundamentación teórica de un artefacto software novedoso que ocupa una brecha crítica en la ingeniería de usabilidad contemporánea. La investigación ha demostrado que es factible y valioso desplazar el análisis de usabilidad hacia las etapas más tempranas del ciclo de vida del software, integrando la detección automatizada de problemas directamente en el entorno creativo de los diseñadores.

El plugin para Figma desarrollado no es solo una herramienta, sino la materialización de un enfoque proactivo para la gestión de la calidad de la experiencia de usuario. Al proporcionar retroalimentación instantánea y accionable sobre *usability smells* estructurales en prototipos estáticos, se ofrece a los equipos de diseño un mecanismo escalable y eficiente para prevenir la acumulación de deuda de UX, alinear las prácticas de diseño con los principios de desarrollo ágil y reducir los costos asociados a la corrección tardía de defectos.

Al embeber diagnósticos automatizados y basados en evidencia directamente en la principal herramienta de diseño, este trabajo contribuye a democratizar el conocimiento de usabilidad, fomenta una cultura de calidad desde la concepción del producto y, en última instancia, sienta las bases para la creación de productos digitales más robustos, eficientes y centrados en el ser humano.

ANEXOS NUEVOS

Anexos

Anexo A.1: Matriz de Trazabilidad Completa

La presente matriz de trazabilidad ofrece una visión consolidada del alcance y la estructura de la investigación, estableciendo una correspondencia explícita entre los objetivos específicos (OE) planteados en el Capítulo 1 y los *usability smells* estructurales que constituyen el objeto de estudio. Este artefacto sirve como un mapa conceptual que no solo documenta el estado actual de la implementación del plugin detector, sino que también delinea la hoja de ruta para el trabajo futuro, proveyendo una justificación clara de las decisiones de priorización tomadas.

La matriz permite verificar la coherencia interna del proyecto, demostrando cómo cada *smell* seleccionado contribuye a la consecución de uno o más objetivos de la tesina. Se detalla el estado de implementación ("Implementado"), la viabilidad técnica estimada para su detección automática ("Viabilidad") y la fase de desarrollo en la que se aborda ("Iteración"). El estado "Actual" corresponde a los detectores completamente implementados y validados en el marco de este trabajo, mientras que "N+1" designa aquellos cuya implementación se propone como línea de trabajo futuro, en consonancia con las conclusiones y la evolución delineada en los Capítulos 7 y 8.[1]

La selección y priorización de los *smells* implementados (S01 a S07) se basó en su alta prevalencia en el diseño de formularios web, la claridad de sus señales estructurales observables en prototipos estáticos de Figma y su impacto directo en dimensiones clave de la usabilidad como la eficiencia, la prevención de errores y la carga cognitiva.[1] Los

smells propuestos para la iteración N+1 (S08 a S11) representan una extensión natural que aborda aspectos más sutiles de la consistencia visual y la claridad semántica, cuya viabilidad de detección es media o condicionada a convenciones de diseño más estrictas.[1]

Tabla A.1.1: Matriz de Trazabilidad de Smells y Objetivos

Smell	Código	Implementado	Objetivos Relevantes	Viabilidad	Iteración
Campo con ancho inadecuado	S01	Sí	OE2, OE3, OE4, OE5, OE6, OE7	Alta	Actual
Inconsistencia de ancho	S02	Sí	OE2, OE3, OE4, OE5, OE7	Alta	Actual
Campo sin formato estructural	S03	Sí	OE2, OE3, OE4, OE5, OE6, OE7	Alta	Actual

Smell	Código	Implementado	Objetivos Relevantes	Viabilidad	Iteración
Vínculo confuso	S04	Sí	OE2, OE3, OE4, OE5, OE7	Alta	Actual
Valor limitado como texto libre	S05	Sí	OE2, OE3, OE4, OE5, OE7	Alta	Actual
Complejidad de formulario	S06	Sí	OE2, OE3, OE4, OE5, OE7	Alta	Actual
Flujo lineal extenso	S07	Sí	OE2, OE3, OE4, OE5, OE7	Alta	Actual
Label ausente o lejano	S08	No	OE2, OE6, OE7	Media	N+1
Desalineación horizontal	S09	No	OE2, OE6, OE7	Media	N+1
Espaciado vertical inconsistente	S10	No	OE2, OE6, OE7	Media	N+1
Placeholder como label	S11	No	OE2, OE6, OE7	Condicionada	N+1

Esta matriz, por lo tanto, no es un mero inventario, sino un instrumento estratégico que articula el qué (los *smells*), el porqué (los objetivos) y el cómo (la implementación y su evolución). Al formalizar estas relaciones, se proporciona al lector una guía clara para navegar la contribución del trabajo y comprender sus límites definidos, alineándose con la metodología de Ingeniería de Artefactos que exige una documentación rigurosa del producto desarrollado y su contexto.[1]

Anexo A.2: Estructura de Archivos del Proyecto y Componentes Arquitectónicos

La arquitectura del plugin de Figma se ha diseñado siguiendo principios de modularidad y separación de conceptos para facilitar su mantenimiento, extensibilidad y comprensión, en cumplimiento con el Objetivo Específico 4 (OE4).[1] La estructura del proyecto refleja esta filosofía, dividiendo la lógica del plugin en componentes bien definidos que se corresponden con las responsabilidades de la aplicación: la interacción con la API de Figma, la ejecución de las heurísticas de detección, y la presentación de los resultados al usuario.

A continuación, se detalla la función de cada archivo y directorio principal dentro del código fuente del proyecto:

- **manifest.json**: Este es el archivo de manifiesto, un requisito indispensable para cualquier plugin de Figma. Define los metadatos esenciales de la extensión, como su nombre (Simple Smells Detector), la versión de la API de Figma que utiliza ("api": "1.0.0"), el punto de entrada para la interfaz de usuario ("ui": "ui.html") y el punto de entrada para la lógica principal que se ejecuta en el *sandbox* de Figma ("main": "code.js"). También especifica los permisos necesarios para que el plugin funcione correctamente.
- **code.ts**: Es el corazón del plugin, donde reside la lógica de *backend*. Este archivo, escrito en TypeScript para mayor robustez y escalabilidad, contiene el hilo principal que se ejecuta en el entorno de Figma. Sus responsabilidades incluyen:
- Interactuar con la API de Figma para acceder al árbol de nodos del documento (figma.currentPage.findAll(...)).
- Orquestar la ejecución de los diferentes módulos detectores.
- Gestionar la comunicación bidireccional con la interfaz de usuario (ui.html) para enviar los hallazgos y recibir comandos del usuario (e.g., iniciar un análisis).
- Implementar el núcleo de análisis y el motor de heurísticas, tal como se describe en la arquitectura del Capítulo 5.[1]
- **ui.html**: Este archivo define la estructura semántica de la interfaz de usuario del plugin, que es la ventana que el diseñador ve e interactúa. Está construido con HTML estándar y contiene los elementos para la visualización de resultados, como las pestañas para filtrar por severidad, los contenedores para la lista de *findings*, y los botones de acción (e.g., "Analizar", "Exportar"). La lógica de la interfaz (manipulación del DOM, gestión de eventos de clic) se maneja a través de un script enlazado.
- **ui.css**: Hoja de estilos en cascada que define la apariencia visual de ui.html. Se encarga de que la interfaz del plugin sea clara, usable y estéticamente coherente con el entorno de Figma, garantizando una experiencia de usuario fluida.
- **/detectors/**: Este directorio materializa el principio de modularidad de la arquitectura. Contiene archivos individuales para cada detector de *smell*, por ejemplo, s01-inadequate-width.ts, s02-inconsistent-width.ts, etc. Cada archivo exporta una función que toma como entrada el conjunto de nodos a analizar y devuelve una lista de hallazgos. Este diseño permite añadir nuevos detectores o modificar los existentes sin afectar el resto del sistema, cumpliendo con el objetivo de extensibilidad (OE6).[1]
- **/utils/**: Directorio que alberga módulos con funciones de utilidad reutilizadas a lo largo de la base de código. Esto evita la duplicación de código y centraliza la lógica común. Ejemplos de utilidades incluyen:
- geometry.ts: Funciones para cálculos geométricos, como la distancia entre nodos o la detección de superposición.
- text.ts: Funciones para la normalización y análisis de contenido textual, utilizadas por detectores como el de vínculos confusos (S04).
- semantics.ts: Lógica para la inferencia de tipos de datos a partir de las etiquetas de los campos (e.g., identificar un campo como "email" o "teléfono"), crucial para el detector S01.

Esta estructura de archivos no solo organiza el código de manera lógica, sino que también proporciona una evidencia concreta de las afirmaciones arquitectónicas realizadas en la tesina. Demuestra la aplicación de buenas prácticas de ingeniería de software y sienta las bases para la futura extensión del artefacto, una de las contribuciones clave de la investigación.[1]

Anexo A.3: Convenciones Detalladas de Severidad de Hallazgos

Para transformar la detección de *usability smells* en una herramienta pragmática y accionable, especialmente en entornos de desarrollo ágil donde la priorización es fundamental, el plugin implementa un sistema de clasificación de severidad. Este sistema asigna a cada hallazgo (*finding*) un nivel de impacto potencial (Alto, Medio o Bajo), guiando al diseñador sobre qué problemas requieren atención inmediata y cuáles pueden ser considerados como mejoras secundarias. La formalización de estos criterios es esencial para garantizar la consistencia y la interpretabilidad de los reportes generados.

La asignación de severidad no es arbitraria, sino que se basa en el impacto estimado del *smell* sobre las dimensiones fundamentales de la usabilidad definidas por normativas como la ISO 9241-11: efectividad, eficiencia y satisfacción.[1] Un problema que puede impedir a un usuario completar una tarea (efectividad) se considera más grave que uno que simplemente ralentiza la interacción (eficiencia).

La siguiente tabla detalla los criterios utilizados para cada nivel de severidad, justificando su impacto y asociándolos con los *smells* específicos implementados en el sistema, tal como se describe en el Capítulo 6.[1]

Tabla A.3.1: Criterios de Clasificación de Severidad

Severidad	Criterio General	Ejemplos de Smells Asociados (Código)	Justificación del Impacto
High (●)	Impacto directo y crítico en la completitud de la tarea, la seguridad de los datos o la comprensión fundamental del flujo. Genera una alta probabilidad de error o abandono.	**S06** (Complejidad de formulario excesiva), **S07** (Flujo lineal extenso)	Estos *smells* representan barreras significativas que pueden impedir o desincentivar fuertemente al usuario a completar una tarea principal. Un formulario con demasiados campos puede abrumar al usuario y llevar a altas tasas de abandono, mientras que un flujo excesivamente largo afecta directamente la eficiencia y puede hacer que el proceso se perciba como inviable.[1]
Medium (◍)	Degradación perceptible de la eficiencia, consistencia o carga cognitiva. No bloquea la tarea, pero introduce fricción, frustración y aumenta la probabilidad de errores menores.	**S02** (Inconsistencia de ancho), **S01** (Campo con ancho inadecuado), **S05** (Valor limitado como texto libre)	Obliga al usuario a realizar un esfuerzo cognitivo o físico adicional, como redimensionar mentalmente la información esperada o escribir manualmente valores que podrían seleccionarse. Afecta la percepción de profesionalismo y la calidad de la interfaz, erosionando la satisfacción del usuario.[1]
Low (○)	Oportunidades de mejora y refinamiento estético o de claridad. No afectan significativamente la tarea principal, pero su corrección mejora la calidad percibida y reduce la micro-fricción.	**S04** (Vínculo confuso), **S03** (Campo sin formato estructural)	Representan "asperezas" en la experiencia que, aunque menores, acumulativamente degradan la satisfacción y la claridad. Un vínculo ambiguo puede causar una momentánea duda, y un campo sin formato puede requerir un pequeño esfuerzo extra de interpretación. Su corrección representa una mejora de bajo costo y alto impacto en la percepción de pulcritud y cuidado del diseño.[1]

Este marco de severidad es el mecanismo que convierte al plugin de un simple "linter" de diseño en un asistente pragmático. Al proporcionar una jerarquía de importancia, la herramienta se alinea con las realidades de los ciclos de desarrollo ágiles, permitiendo a los equipos de diseño tomar decisiones informadas sobre cómo invertir su tiempo para maximizar el valor entregado al usuario en cada iteración.[1]

Anexo A.4: Ejemplo de Reporte de Análisis Exportado (Formato Markdown)

Una de las funcionalidades clave del plugin, en línea con el objetivo OE6, es la capacidad de exportar los resultados del análisis a un formato portátil y legible por humanos, como Markdown.[1] Esto permite integrar los hallazgos en sistemas de gestión de proyectos (como Jira, Trello o Azure DevOps), documentación (como Confluence o Notion) o simplemente compartirlos en reportes.

El siguiente es un ejemplo completo de un reporte exportado. El formato está diseñado para ser a la vez conciso y exhaustivo, proporcionando un resumen ejecutivo, métricas por detector y un listado detallado de cada hallazgo, incluyendo su severidad, descripción, sugerencia de mejora y metadatos para su localización precisa en el archivo de Figma.

Reporte del Verificador de Smells - Simple Smells Detector

Fecha de Análisis: 29 de septiembre de 2025, 14:30

Archivo Analizado: Onboarding_v3.fig

Alcance del Análisis: Página "Registro de Usuario"

Resumen Ejecutivo

\

- **Total de Hallazgos:** 12
- **Hallazgos por Severidad:**
- 🔘 **High:** 2
- ⚪ **Medium:** 7
- 🔘 **Low:** 3
- **Tiempo Total de Análisis:** 312ms
- **Nodos Analizados:** 458

Métricas por Detector

Detector (Código)	Hallazgos	Tiempo (ms)	Estado
S01: Campo con ancho inadecuado	4	45	✅
S02: Inconsistencia de ancho	1	88	✅
S03: Campo sin formato estructural	2	35	✅
S04: Vínculo confuso	1	21	✅
S05: Valor limitado como texto libre	1	41	✅
S06: Complejidad de formulario	1	75	✅
S07: Flujo lineal extenso	2	7	✅

Listado Detallado de Hallazgos

🔘 **Severidad Alta**

\

1. **ID del Nodo:** 105:2

- **Nombre del Nodo:** Formulario de Registro Completo
- **Smell:** Complejidad de formulario (S06)
- **Descripción:** El grupo de campos identificado contiene 14 inputs, superando el umbral recomendado de 10. Esto puede generar una alta carga cognitiva y aumentar la tasa de abandono.
- **Sugerencia:** Considere dividir el formulario en varios pasos (multi-step wizard) o posponer la solicitud de información no esencial para después del registro inicial.

2. **ID del Nodo:** 110:5

- **Nombre del Nodo:** Flujo de Verificación
- **Smell:** Flujo lineal extenso (S07)
- **Descripción:** El flujo de prototipo "Verificación de Cuenta" contiene 7 pantallas consecutivas sin ramificaciones, lo que puede resultar tedioso para el usuario.
- **Sugerencia:** Evalúe si es posible combinar algunos pasos o simplificar el proceso de verificación para reducir la longitud del flujo.

⚪ **Severidad Media**

\

3. **ID del Nodo:** 105:15

- **Nombre del Nodo:** Input_Email
- **Smell:** Campo con ancho inadecuado (S01)

- **Descripción:** El campo "Email" tiene un ancho de 150px, que es inferior al mínimo recomendado de 200px para este tipo de dato. Puede no ser suficiente para visualizar direcciones de correo largas.
- **Sugerencia:** Ajuste el ancho del campo a un valor entre 200px y 350px para mejorar la legibilidad.

4. **ID del Nodo:** 105:22

- **Nombre del Nodo:** Input_País
- **Smell:** Valor limitado como texto libre (S05)
- **Descripción:** El campo "País" es un input de texto libre. Este tipo de dato proviene de un vocabulario limitado y conocido.
- **Sugerencia:** Reemplace el campo de texto por un componente selector (dropdown) con funcionalidad de búsqueda o autocompletado para reducir errores y facilitar la entrada.

... (otros hallazgos de severidad media y baja)...

Anexo A.5: Catálogo Extendido de Usability Smells Estructurales

Este anexo presenta un catálogo detallado de los *usability smells* estructurales identificados en esta investigación, tanto los implementados (S01-S07) como los propuestos para trabajo futuro (S08-S11). Cada entrada del catálogo proporciona una descripción formal del *smell*, las señales observables en un prototipo estático que permiten su detección, una justificación de su impacto en la usabilidad fundamentada en la literatura, y las refactorizaciones de UX sugeridas para su corrección. Este catálogo consolida el conocimiento disperso a lo largo de la tesina y lo enriquece con referencias a trabajos seminales en el campo.[1]

S01: Campo con Ancho Inadecuado (Inadequate Field Width)

\

- **Descripción:** Un campo de entrada de texto (input) presenta un ancho visual que no se corresponde con la longitud esperada del dato que debe contener. Puede ser demasiado corto, ocultando parte del contenido y dificultando su revisión, o excesivamente largo, rompiendo la armonía visual del formulario y generando expectativas incorrectas sobre la longitud del dato.[1]
- **Señales Estructurales:**

1. **Geometría:** El ancho (width) del nodo rectangular que representa el campo de entrada.
2. **Texto Asociado:** El contenido textual del nodo Text más cercano (la *label*), que permite inferir semánticamente el tipo de dato esperado (e.g., "Email", "Teléfono", "Código Postal").
3. **Taxonomía Interna:** Un mapeo predefinido que asocia tipos de datos inferidos con rangos de ancho mínimo y máximo recomendados (e.g., Email: min 200px, max 350px).[1]

- **Justificación de Usabilidad:** Este *smell* viola la heurística de "Correspondencia entre el sistema y el mundo real" de Nielsen, ya que el affordance visual del campo no se alinea con las expectativas del usuario sobre el dato a ingresar.[1] Un campo corto para un email largo genera frustración y aumenta la probabilidad de errores de tipeo no visibles. Afecta la **eficiencia** (requiere desplazamiento horizontal) y la **prevención de errores**.[1]
- **Refactorizaciones Sugeridas:**
- **Ajustar Dimensiones (Resize Element):** Modificar el ancho del campo para que se ajuste al rango óptimo para el tipo de dato inferido.[1]
- **Cambiar Widget (Change Widget):** Si el dato es muy largo, considerar reemplazar un input de una línea por un textarea de múltiples líneas (e.g., para un campo "Descripción").[1]

S02: Inconsistencia de Ancho (Width Inconsistency)

\

- **Descripción:** Dentro de un mismo formulario o grupo lógico de campos, los anchos de los inputs varían sin una justificación funcional o semántica aparente. Esta falta de alineación vertical crea un desorden visual que dificulta el escaneo rápido del formulario.[1]
- **Señales Estructurales:**

1. **Proximidad Espacial:** Un conjunto de nodos de tipo input agrupados lógicamente, identificados mediante algoritmos de clustering basados en la distancia vertical y horizontal entre ellos.[1]
2. **Geometría:** La colección de anchos de todos los campos dentro del grupo identificado.
3. **Análisis Estadístico:** Una desviación estándar de los anchos que supera un umbral predefinido, indicando una variabilidad significativa.

- **Justificación de Usabilidad:** Atenta contra la "Consistencia y estándares" y la "Estética y diseño minimalista" de Nielsen.[1] Un layout desordenado aumenta la carga cognitiva, ya que el ojo no puede seguir una línea vertical clara al pasar de un campo al siguiente. Esto ralentiza el proceso de llenado del formulario, impactando negativamente la **eficiencia** y la **satisfacción** del usuario.[1]
- **Refactorizaciones Sugeridas:**
- **Normalizar Dimensiones:** Estandarizar el ancho de todos los campos del grupo a un valor común (e.g., el promedio, el máximo, o un valor definido en el sistema de diseño).

S03: Campo sin Formato Estructural (Unformatted Input)

\

- **Descripción:** Se utiliza un campo de texto libre para solicitar datos que poseen una estructura interna bien definida y esperada, como fechas (DD/MM/AAAA), números de tarjeta de crédito (XXXX-XXXX-XXXX-XXXX) o CUIT/CUIL (XX-XXXXXXXX-X), sin proveer una máscara de entrada o componentes que guíen al usuario.[1]
- **Señales Estructurales:**

1. **Texto Asociado:** La etiqueta del campo contiene palabras clave como "Fecha", "Vencimiento", "Teléfono", "CUIT", etc.
2. **Ausencia de Componente:** El prototipo no utiliza un componente específico (como un Datepicker o un grupo de selects) para este tipo de dato, sino un input genérico.

- **Justificación de Usabilidad:** Es una causa principal de errores de validación. La ambigüedad sobre el formato esperado (e.g., ¿01/12/2025 o 12/01/2025?) viola la "Prevención de errores".[1] Obliga al usuario a un proceso de "adivinación" que incrementa la carga cognitiva y la frustración, afectando la **efectividad** y la **eficiencia**.[1]
- **Refactorizaciones Sugeridas:**
- **Añadir Máscara de Formato (Format Input):** Aplicar una máscara visual que guíe la entrada de datos.[1]
- **Añadir Datepicker (Add Datepicker):** Para fechas, incorporar un widget de calendario.[1]
- **Convertir Input en Selects (Date Input into Selects):** Para fechas, reemplazar el campo por tres menús desplegables (día, mes, año).[1]

S04: Vínculo Confuso (Misleading Link)

\

- **Descripción:** Un enlace o botón utiliza un texto genérico y no descriptivo que no informa adecuadamente al usuario sobre su destino o la acción que se ejecutará al hacer clic. Ejemplos clásicos son "Clic aquí", "Ver más", "Leer más".[1]
- **Señales Estructurales:**

1. **Tipo de Nodo:** El nodo es un enlace de prototipo de Figma.
2. **Contenido Textual:** El texto del enlace, una vez normalizado (minúsculas, sin acentos), coincide con una lista predefinida de frases ambiguas.
3. **Análisis de Longitud:** Textos de enlace excesivamente cortos (e.g., una o dos palabras genéricas).

- **Justificación de Usabilidad:** Viola la "Visibilidad del estado del sistema" y la "Correspondencia con el mundo real".[1] El usuario no puede predecir el resultado de su acción, lo que genera incertidumbre y puede llevar a navegaciones innecesarias ("pogo-sticking"). Es también un problema de **accesibilidad**, ya que los lectores de pantalla leen estos enlaces fuera de contexto, resultando en una experiencia de navegación incomprensible.[1]
- **Refactorizaciones Sugeridas:**
- **Renombrar Elemento (Rename Element):** Modificar el texto del enlace para que sea descriptivo y único, indicando la acción o el destino. Por ejemplo, cambiar "Ver más" por "Ver detalles del producto".[1]

S05: Valor Limitado como Texto Libre (Free Input for Limited Values)

\

- **Descripción:** Se utiliza un campo de texto libre para datos que deben seleccionarse de un conjunto de opciones predefinido, finito y generalmente pequeño (e.g., país, género, tipo de documento, estado civil).[1]
- **Señales Estructurales:**

1. **Texto Asociado:** La etiqueta del campo contiene palabras clave que sugieren un conjunto cerrado de opciones, como "País", "Provincia", "Género", "Categoría".
2. **Tipo de Nodo:** Es un input de texto estándar.

- **Justificación de Usabilidad:** Introduce una alta probabilidad de errores de entrada (e.g., "Argentina", "arg", "Republica Argentina") y inconsistencia de datos. Viola la "Prevención de errores" y el principio de "Reconocimiento sobre recuerdo", ya que obliga al usuario a recordar y tipear las opciones en lugar de simplemente seleccionarlas.[1] Afecta negativamente la **efectividad** y la **eficiencia**.[1]
- **Refactorizaciones Sugeridas:**
- **Convertir Input en Select (Turn Input into Select):** Reemplazar el campo de texto por un menú desplegable, idealmente con búsqueda, si la lista es larga.[1]
- **Convertir Input en Radio Buttons (Turn Input into Radios):** Si el número de opciones es pequeño (típicamente 2-5), usar botones de opción para mostrar todas las alternativas simultáneamente.[1]
- **Añadir Autocompletado (Add Autocomplete):** Si la lista es muy extensa pero hay valores populares, añadir una funcionalidad de autocompletado puede ser una solución intermedia.[1]

S06: Complejidad de Formulario (Form Complexity)

\

- **Descripción:** Un único formulario o pantalla solicita una cantidad excesiva de información al usuario de una sola vez. No existe un número mágico, pero formularios con más de 10-12 campos suelen percibirse como abrumadores y disuaden al usuario de completarlos.[1]
- **Señales Estructurales:**

1. **Proximidad Espacial:** Un grupo de campos de entrada identificados como un único formulario mediante algoritmos de clustering.[1]
2. **Recuento de Nodos:** El número de inputs, selects, textareas y otros controles interactivos dentro del grupo supera un umbral configurable (e.g., 10).

- **Justificación de Usabilidad:** Genera una alta **carga cognitiva** y puede llevar directamente al *smell* de comportamiento "Formulario Abandonado" (*Abandoned Form*).[1] Viola el principio de "Estética y diseño minimalista" al presentar más información de la necesaria en un solo paso. Afecta directamente la **efectividad** (tasa de completitud) y la **satisfacción**.[1]
- **Refactorizaciones Sugeridas:**
- **Dividir Página/Actividad (Split Page / Split Activity):** Reestructurar el formulario en un proceso de varios pasos (wizard), agrupando los campos por afinidad temática en cada paso.[1]
- **Posponer Actividad (Postpone Activity):** Evaluar qué campos son absolutamente esenciales para el paso actual y mover la solicitud de información secundaria a una etapa posterior del ciclo de vida del usuario (e.g., en la configuración de su perfil).[1]

S07: Flujo Lineal Extenso (Long Linear Flow)

\

- **Descripción:** Un flujo de prototipo interactivo define una secuencia larga de pantallas que el usuario debe atravesar de manera estrictamente lineal para completar una tarea, sin ramificaciones ni atajos.
- **Señales Estructurales:**

1. **Metadatos de Prototipado:** Análisis de las conexiones (prototype connections) entre *frames* en Figma.
2. **Análisis de Grafo:** Identificación de una ruta en el grafo de prototipo que consiste en una cadena de N nodos (N > umbral, e.g., 5-7) sin nodos de salida intermedios o rutas alternativas.

- **Justificación de Usabilidad:** Flujos largos y rígidos pueden ser tediosos y frustrantes, afectando la **eficiencia**. Violan el principio de "Flexibilidad y eficiencia de uso", que sugiere la existencia de aceleradores para usuarios expertos, y el de "Control y libertad del usuario", al no permitirle navegar libremente.[1]
- **Refactorizaciones Sugeridas:**
- **Añadir Atajo/Enlace (Add Link):** Incorporar enlaces que permitan a los usuarios saltar pasos no esenciales o navegar directamente a secciones de interés.[1]
- **Fusionar Páginas (Merge Pages):** Combinar pasos o pantallas que son conceptualmente simples y pueden presentarse juntas para reducir el número total de clics.[1]

S08 a S11: Smells Propuestos para Trabajo Futuro (N+1)

\

- **S08: Label Ausente o Lejano (Missing or Distant Label):** Un campo de entrada no tiene una etiqueta textual visible o esta se encuentra a una distancia tal que su asociación visual es ambigua. Impacta la **comprensión** y la **accesibilidad**.
- **S09: Desalineación Horizontal (Horizontal Misalignment):** Elementos que deberían estar alineados horizontalmente (e.g., un label y su input) presentan una desviación notable en su eje Y, rompiendo la línea de lectura. Afecta la **estética** y la **eficiencia de escaneo**.
- **S10: Espaciado Vertical Inconsistente (Inconsistent Vertical Spacing):** El espacio vertical entre pares de label-input o entre diferentes secciones de un formulario es irregular, dificultando la percepción de grupos lógicos. Afecta la **carga cognitiva** y la **consistencia**.
- **S11: Placeholder como Label (Placeholder as Label):** Se utiliza el atributo placeholder de un campo de entrada como su única etiqueta. La etiqueta desaparece una vez que el usuario comienza a escribir, eliminando el contexto y violando el principio de "Reconocimiento sobre recuerdo". Es un problema grave de **usabilidad** y **accesibilidad**.

Anexo A.6: Plantilla Estructurada para Dataset de Validación Empírica

Para avanzar hacia la validación empírica y la integración de técnicas de aprendizaje automático, como se discute en el Capítulo 8, es fundamental la creación de un dataset de alta calidad.[1] Este anexo propone una plantilla o esquema en formato JSON para la anotación manual de componentes de Figma por parte de expertos en UX. El objetivo es construir un corpus de "ground truth" que relacione artefactos de diseño con la presencia (o ausencia) de

usability smells.

Este dataset permitiría entrenar y validar modelos predictivos, medir con precisión el rendimiento (precisión, exhaustividad) de los detectores heurísticos actuales y explorar la detección de *smells* más complejos mediante aprendizaje supervisado.

Esquema Propuesto (Annotation.json):

JSON

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Anotación de Usability Smell para un Nodo de Figma",
  "type": "object",
  "properties": {
    "fileId": {
      "description": "Identificador único del archivo de Figma.",
      "type": "string"
    },
    "nodeId": {
      "description": "Identificador único del nodo dentro del archivo de Figma.",
      "type": "string"
    },
    "nodeType": {
      "description": "Tipo de nodo según la API de Figma (e.g., RECTANGLE, TEXT, FRAME).",
      "type": "string"
    },
    "expertAnnotations": {
      "description": "Lista de anotaciones realizadas por expertos en UX.",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "expertId": {
            "description": "Identificador anónimo del experto que realiza la anotación.",
            "type": "string"
          },
          "timestamp": {
            "description": "Fecha y hora de la anotación en formato ISO 8601.",
            "type": "string",
```

```
                "format": "date-time"
            },
            "detectedSmells": {
                "description": "Lista de smells identificados por el experto en este nodo.",
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "smellCode": {
                            "description": "Código del smell (e.g., 'S01', 'S06').",
                            "type": "string"
                        },
                        "confidence": {
                            "description": "Confianza del experto en la presencia del smell (1=Baja, 5=Muy Alta).",
                            "type": "integer",
                            "minimum": 1,
                            "maximum": 5
                        },
                        "justification": {
                            "description": "Comentario cualitativo del experto explicando su razonamiento.",
                            "type": "string"
                        }
                    },
                    "required": ["smellCode", "confidence"]
                }
            },
            "isFalsePositiveOf": {
                "description": "Si el plugin detectó un smell que el experto considera incorrecto, se anota aquí.",
                "type": "string",
                "pattern": "^S[0-9]{2}$"
            }
        },
        "required": ["expertId", "timestamp"]
    }
  }
},
"required": ["fileId", "nodeId", "expertAnnotations"]
}
```

Uso y Justificación:

- **fileId y nodeId**: Permiten vincular inequívocamente cada anotación con un elemento específico en un diseño de Figma, garantizando la trazabilidad.
- **expertAnnotations**: La estructura de array permite recoger juicios de múltiples expertos sobre el mismo nodo, lo que es crucial para medir la concordancia inter-evaluador (e.g., usando el coeficiente Kappa de Fleiss) y construir un dataset más robusto.[1]
- **detectedSmells**: Permite que un experto identifique múltiples *smells* en un mismo componente. La confidence captura la subjetividad inherente a la evaluación heurística, mientras que la justification provee datos cualitativos valiosos.
- **isFalsePositiveOf**: Este campo es fundamental para el re-entrenamiento y calibración de los detectores existentes. Permite registrar explícitamente los casos en que la herramienta se equivoca, proveyendo un feedback directo para su mejora.

La adopción de este esquema estandarizado es el primer paso tangible hacia la validación empírica rigurosa y la evolución del sistema hacia un asistente de diseño más inteligente y preciso.

Anexo A.7: Hoja de Ruta Técnica y de Investigación (Trabajo Futuro)

Este anexo sistematiza y expande las líneas de trabajo futuro delineadas en los Capítulos 7 y 8, presentando una hoja de ruta estructurada para la evolución del artefacto y la investigación asociada. Se dividen las futuras acciones en tres ejes estratégicos: expansión de la cobertura de detectores, evolución hacia un sistema prescriptivo y de cuantificación, y validación empírica rigurosa.

1. Expansión de la Cobertura de Detectores

La capacidad actual del plugin se centra en un conjunto de *smells* estructurales de alta prevalencia. La siguiente fase de desarrollo se enfocará en ampliar esta cobertura para abordar nuevas categorías de problemas de usabilidad y accesibilidad.

- **Detectores de Consistencia de Layout:**
- **Acción:** Implementar los detectores propuestos S08 (Label ausente/lejano), S09 (Desalineación horizontal) y S10 (Espaciado vertical inconsistente).
- **Justificación:** Estos *smells* afectan la legibilidad y la carga cognitiva. Su detección automática reforzaría la capacidad del plugin para actuar como un guardián de la consistencia del sistema de diseño.
- **Detectores de Accesibilidad (WCAG):**
- **Acción:** Desarrollar un nuevo conjunto de detectores enfocados en criterios de accesibilidad básicos que puedan ser evaluados estáticamente, como el contraste de color entre texto y fondo, y el uso de placeholders como única etiqueta (S11).
- **Justificación:** La accesibilidad es una dimensión fundamental de la usabilidad.[1] Integrar estas verificaciones tempranamente en el flujo de diseño es una extensión de alto impacto que responde a una necesidad crítica de la industria.
- **Integración de Modelos de IA (Análisis Semántico y Visual):**
- **Acción:** Investigar y prototipar la integración de modelos de Procesamiento de Lenguaje Natural (PLN) para analizar la claridad y consistencia semántica de las etiquetas y textos, permitiendo una detección más inteligente de *smells* como "Vínculo Confuso" (S04).[1]
- **Acción:** Explorar el uso de modelos de visión por computadora para identificar problemas de desorden visual (*visual clutter*) o inconsistencias estilísticas que las heurísticas geométricas simples no pueden capturar.
- **Justificación:** La IA permitiría al plugin superar las limitaciones del análisis puramente estructural y abordar una clase de *smells* semánticos y contextuales, aumentando significativamente su inteligencia y utilidad.[1]

2. Hacia la Sugerencia de Refactorizaciones y la Cuantificación del Impacto

La evolución más significativa del plugin será su transición de una herramienta puramente diagnóstica a una prescriptiva, que no solo identifica problemas sino que también sugiere soluciones concretas y cuantifica su posible impacto.

- **Sugerencia de Refactorizaciones de UX:**
- **Acción:** Para cada *smell* detectado, el plugin deberá sugerir una o más refactorizaciones de UX catalogadas como posibles soluciones.[1] Por ejemplo, ante un *smell* S05 (*Free Input for Limited Values*), se propondrán las refactorizaciones "Turn Input into Select" o "Turn Input into Radios".[1]
- **Justificación:** Esto cierra el ciclo de detección-corrección, haciendo que la herramienta sea mucho más accionable y educativa para el diseñador.
- **Integración de Métricas Predictivas (Interaction Effort):**
- **Acción:** Integrar un modelo predictivo basado en métricas como el *Interaction Effort*.[1] Al sugerir refactorizaciones, el plugin podría ofrecer una estimación cuantitativa de la reducción del esfuerzo de interacción que cada opción aportaría.
- **Justificación:** Esto proporciona una base objetiva para que el diseñador elija entre múltiples soluciones, transformando la toma de decisiones de un proceso intuitivo a uno basado en datos predictivos.[1]

3. Validación Empírica y Estudios de Campo

Para establecer con rigor científico el valor y el impacto del plugin en la práctica profesional, es imperativo realizar estudios de validación empírica.

- **Creación de un Dataset Anotado:**
- **Acción:** Utilizar la plantilla del Anexo A.6 para llevar a cabo un proceso de anotación con múltiples expertos en UX sobre un conjunto diverso de prototipos de Figma.
- **Justificación:** Este dataset público y anonimizado será la piedra angular para medir la precisión y exhaustividad de los detectores actuales y futuros, y para entrenar modelos de aprendizaje automático.[1]
- **Experimento Controlado (A/B Testing):**
- **Acción:** Diseñar y ejecutar un estudio controlado donde un grupo de diseñadores (grupo de tratamiento) utilice el plugin para realizar una tarea de diseño, mientras que un grupo de control no lo hace. Se medirán métricas como el tiempo de tarea, el número de *smells* residuales en el diseño final y la calidad percibida del resultado.
- **Justificación:** Este tipo de estudio proporcionará evidencia cuantitativa del impacto del plugin en la eficiencia del proceso de diseño y en la calidad del producto final, validando empíricamente las hipótesis centrales de la tesina.[1]

Esta hoja de ruta define una trayectoria clara para que el proyecto evolucione desde un artefacto de investigación robusto a una herramienta madura y validada, con el potencial de generar un impacto significativo en las prácticas de diseño de UX.

Anexo A.8: Bibliografía Completa

Esta sección presenta la lista completa de referencias citadas a lo largo de la tesina, formateadas de acuerdo con la 7ª edición del Manual de Publicaciones de la American Psychological Association (APA). La compilación se ha realizado a partir de una revisión exhaustiva del cuerpo del documento y los materiales de investigación de soporte.[2]

Almeida, D., Saraiva, J., Campos, J. C., & Silva, J. C. (2015). Towards a Catalog of Usability Smells. En *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (pp. 1211-1218). ACM. [1]

Baltes, S., & Domaschka, J. (2021). Understanding the 'What' and the 'How' of UX Debt: A Mixed-Methods Study on the Definition and Management of Usability and User Experience Issues. En *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1-15). ACM. [1]

Bargas-Avila, J. A., & Hornbæk, K. (2011). Old wine in new bottles or novel challenges: A critical analysis of empirical studies of user experience. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2689–2698). ACM. [1]

Beck, K. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addison-Wesley Professional. [1]

Cunningham, W. (1992). The WyCash portfolio management system. *OOPSLA '92 Addendum to the Proceedings*, 29–30. [1]

Da Silva, T., Martin, A., Maurer, F., & Silveira, M. (2011). User-centered design and agile methods: A systematic review. En *2011 AGILE Conference* (pp. 77-86). IEEE. [1]

DaFonseca, R. S., de Souza, C. R., & de Souza, S. R. S. (2019). A Preliminary Model for UX Debt Management. En *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1-6). IEEE. [1]

Fernandez, A., Insfran, E., & Abrahão, S. (2011). Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology*, *53*(8), 789-817. [1]

Gardey, J. C., Garrido, A., Firmenich, S., Grigera, J., & Rossi, G. (2020). UX-Painter: An Approach to Explore Interaction Fixes in the Browser. *Proceedings of the ACM on Human-Computer Interaction*, *4*(EICS), 1-21. [1]

Gardey, J. C., Grigera, J., Rodríguez, A., Rossi, G., & Garrido, A. (2021). Predicting Interaction Effort in Web Interface Widgets. *ArXiv Preprint arXiv:2107.08119*. [1]

Garrido, A., Rossi, G., & Distante, D. (2010). Refactoring for usability in web applications. En *Proceedings of the 2010 ICSE Workshop on Refactoring Tools* (pp. 59-62). ACM. [1]

Garrido, A., Rossi, G., Medina-Medina, N., Grigera, J., & Firmenich, S. (2014). Improving Accessibility of Web Interfaces: Refactoring to the Rescue. *Universal Access in the Information Society*, *13*(4), 435-451. [1]

Grigera, J. (2017). *Self-Refactoring: mejoras automáticas de usabilidad para aplicaciones web*. Universidad Nacional de La Plata. [1]

Grigera, J., Garrido, A., & Rivero, J. M. (2014). A Tool for Detecting Bad Usability Smells in an Automatic Way. En *Web Engineering. ICWE 2014 International Workshops* (pp. 490-493). Springer. [1]

Grigera, J., Garrido, A., Rivero, J. M., & Rossi, G. (2017). Automatic Detection of Usability Smells in Web Applications. *Journal of Web Engineering*, *16*(3&4), 296-331. [1]

Grigera, J., Gardey, J. C., Rodriguez, A., Garrido, A., & Rossi, G. (2019). One Metric for All: Calculating Interaction Effort of Individual Widgets. En *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI' 19 Extended Abstracts)* (pp. 1-6). ACM. [1]

Grigera, J., Liste, T., Rossi, G., & Garrido, A. (2021). A Collaborative Tool for Early Usability Testing. En *2021 IEEE/ACM 1st Workshop on Agile Transformation (AT)* (pp. 1-8). IEEE. [1]

Harms, P., & Grabowski, J. (2014). Usage-Based Automatic Detection of Usability Smells. En *Human-Centered Software Engineering* (pp. 217-234). Springer. [1]

Hornbæk, K. (2006). Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, *64*(2), 79-102. [1]

International Organization for Standardization. (2018). *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts (ISO 9241-11:2018)*. [1]

International Organization for Standardization. (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (ISO/IEC 25010:2011)*. [1]

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153-163. [1]

Nguyen, H. V., Nguyen, H. A., Nguyen, A. T., Nguyen, T. T., & Nguyen, T. N. (2012). Detection of Embedded Code Smells in Dynamic Web Applications. En *Proceedings of the 2012 International Conference on Software Engineering* (pp. 557-567). IEEE. [1]

Nielsen, J., & Loranger, H. (2006). *Prioritizing Web Usability*. New Riders. [1]

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. En *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 249-256). ACM. [1]

Norman, D. (2013). *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books. [1]

Paternò, F., Schiavone, A. G., & Conte, A. (2017). Customizable Automatic Detection of Bad Usability Smells in Mobile Accessed Web Applications. En *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services* (pp. 1-12). ACM. [1]

Preece, J., Sharp, H., & Rogers, Y. (2015). *Interaction Design: Beyond Human-Computer Interaction* (4th ed.). Wiley. [1]

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach* (7th ed.). McGraw-Hill. [1]

Rossi, G., Garrido, A., & Distante, D. (2011). Refactoring for Usability in Web Applications. *IEEE Software*, *28*(3), 60-66. [1]

Rubin, J., & Chisnell, D. (2008). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests* (2nd ed.). Wiley. [1]

Sauer, J., & Sonderegger, A. (2022). The influence of aesthetics on usability and hedonics: A moderated mediation model. *Behaviour & Information Technology*, *41*(1), 1-17. [1]

Shi, W., Wang, H., & Xu, Y. (2022). Deep learning for automated usability evaluation of user interfaces. *International Journal of Human-Computer Studies*, *165*, 102874. [1]

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson. [1]

Zhang, L., Xu, Z., & Zhou, M. (2021). AI-Driven Usability Analysis of Static Prototypes: A Systematic Review. *Journal of Usability Studies*, *16*(4), 208–223. [1]

Anexo A.9: Glosario de Términos Fundamentales

Este glosario define los términos clave utilizados en la tesina para asegurar una comprensión unívoca y precisa de los conceptos fundamentales. Las definiciones se basan en la literatura académica de referencia citada a lo largo del documento.

- Deuda de UX (UX Debt):
  Metáfora análoga a la "deuda técnica". Se refiere al conjunto de decisiones de diseño subóptimas o aplazadas que, aunque pueden acelerar la entrega a corto plazo, generan costos acumulativos a largo plazo. Esta deuda se manifiesta en forma de una experiencia de usuario degradada, mayor necesidad de soporte, retrabajo futuro y dificultad para evolucionar el producto.1 La detección temprana de *usability smells* es una estrategia para prevenir la acumulación de esta deuda.[1]
- Detector:
  Módulo de software heurístico diseñado para inspeccionar los metadatos estructurales (geometría, texto, nomenclatura, relaciones) de un prototipo de interfaz en Figma. Su función es identificar y reportar la presencia de un usability smell específico, generando un hallazgo (finding) clasificado con una severidad.1
- Esfuerzo de Interacción (Interaction Effort):
  Métrica unificada propuesta para cuantificar el coste cognitivo y físico que un usuario debe invertir para interactuar con un componente de interfaz (widget) específico. Se calcula a partir de micro-medidas observables en los logs de interacción (e.g., tiempo de permanencia, longitud de la traza del mouse, número de clics) y sirve como un indicador para comparar la eficiencia de diferentes alternativas de diseño.1
- Heurística:
  En el contexto de esta tesina, una regla o principio guía que encapsula una buena práctica de usabilidad. Las heurísticas de Nielsen & Molich (1990) son un ejemplo canónico.1 Los detectores del plugin implementan heurísticas automatizadas para identificar patrones que violan estos principios (e.g., una heurística para la complejidad de un formulario se basa en el recuento de sus campos).

- Prototipo de Baja a Media Fidelidad (Low-to-Mid Fidelity Prototype):
  Representaciones de una interfaz de usuario que se enfocan en la estructura, el flujo y la funcionalidad básica, en lugar de en el detalle visual y estético final. Los prototipos de baja fidelidad (Lo-Fi) suelen ser esquemáticos (wireframes), mientras que los de media fidelidad (Mid-Fi) incorporan mayor detalle en la jerarquía visual y tipografía, pero sin llegar a ser una representación pixel-perfect del producto final.1 El plugin está diseñado para operar sobre este tipo de artefactos.
- Refactorización de UX (UX Refactoring):
  Conjunto de transformaciones aplicadas a una interfaz de usuario con el objetivo de mejorar su calidad de uso (e.g., usabilidad, accesibilidad, eficiencia) sin alterar su funcionalidad esencial. Es una técnica para "pagar" la deuda de UX. Un usability smell actúa como el detonante que sugiere la necesidad de una refactorización.1
- Severidad (Severity):
  Estimación cualitativa (Alta, Media, Baja) del impacto potencial que un usability smell detectado puede tener en la experiencia del usuario. Se utiliza para priorizar los hallazgos, permitiendo a los equipos de diseño enfocarse primero en los problemas más críticos.1
- Usability Smell / UX Smell:
  Un indicio observable en un artefacto de diseño (en este caso, un prototipo) que sugiere un problema potencial de usabilidad o una deficiencia en la experiencia del usuario. Análogo a los code smells en ingeniería de software, un smell no es un error en sí mismo, sino una señal de alerta que justifica una investigación y una posible refactorización.1

Works cited

1. Catálogo de Usability Smells 2.pdf
2. Normas APA 7ª edición - La Salle Pachuca, accessed September 30, 2025, https://www.lasallep.edu.mx/ulsap/Repositorio/guiaAPA/Guía_Normas_APA_7.pdf
3. Estilo APA - Biblioteca, accessed September 30, 2025, https://biblioteca.uoc.edu/es/pagina/Estilo-APA/
4. Normas APA, 7ª edición ANEXO II: FORMATO DE PRESENTACIÓN DE TRABAJOS, accessed September 30, 2025, https://www.ehu.eus/documents/d/biblioteka/estilo-apa-anexo-2-formato-de-presentacion-de-trabajos_
5. Normas APA con plantilla y generador 2025 - Séptima edición, accessed September 30, 2025, https://normasapa.in/
6. Manual de APA 7ª Edición (American Psychological Association) Nuevas Normas de Redacción, Citas y Referencias - Caribbean University, accessed September 30, 2025, https://www.caribbean.edu/Base_de_datos/Nuevas_Normas_del_Manual_APA7.pdf
7. Citar Libro – Referencia Bibliográfica - Normas APA, accessed September 30, 2025, https://normas-apa.org/referencias/citar-libro/
8. APA: cómo citar un libro [Versión 2025] - BibGuru Guides, accessed September 30, 2025, https://www.bibguru.com/es/g/cita-apa-de-libro/
9. HOW TO WRITE A BIBLIOGRAPHY OR REFERENCE IN WORD ACCORDING TO APA STANDARDS, SEVENTH EDITION (7th... - YouTube, accessed September 30, 2025, https://www.youtube.com/watch?v=HZzIcjXoHjM
10. Formato APA con el Generador APA de Scribbr, accessed September 30, 2025, https://www.scribbr.es/citar/generador/apa/
11. Citar Revista – Referencia Bibliográfica - Normas APA, accessed September 30, 2025, https://normas-apa.org/referencias/citar-revista/
12. Tips APA 7 ¿Cómo citar artículos de revistas con un solo autor? - YouTube, accessed September 30, 2025, https://www.youtube.com/watch?v=TXg--N2ZCao
13. APA: cómo citar un artículo de revista científica [Versión 2025] - BibGuru Guides, accessed September 30, 2025, https://www.bibguru.com/es/g/cita-apa-articulo-de-revista-cientifica/
14. NORMAS APA PARA AUTORES DE LA REVISTA, accessed September 30, 2025, http://www.indteca.com/ojs/index.php/Revista_Scientific/libraryFiles/downloadPublic/58
15. Normas APA 7.ª edición. Guía de citación y referenciación, accessed September 30, 2025, https://www.revista.unam.mx/wp-content/uploads/3_Normas-APA-7-ed-2019-11-6.pdf
16. Guía Normas APA, accessed September 30, 2025, https://normas-apa.org/wp-content/uploads/Guia-Normas-APA-7ma-edicion.pdf
17. Cómo Citar Tesis Doctorales y Disertaciones en Normas APA 7ma Edición | Guía Completa, accessed September 30, 2025, https://editorial.excedinter.com/citar-tesis-doctorales-apa-7ma/
18. Citar Tesis o Disertaciones – Referencia Bibliográfica - Normas APA, accessed September 30, 2025, https://normas-apa.org/referencias/citar-tesis-disertaciones/
19. GUÍA RESUMIDA APA 7° EDICIÓN - Universidad Manuela Beltrán, accessed September 30, 2025, https://umb.edu.co/wp-content/uploads/2024/03/guia-resumida-apa-septima-edicion-2024.pdf
20. Cómo Citar Tesis Doctorales y de Maestría en APA 7 con Word | Guía Completa - YouTube, accessed September 30, 2025, https://www.youtube.com/watch?v=hvz8UeTDiJw

HIU MANUAL DE FORMATO DE TESIS DOCTORALES - Humboldt International University, accessed September 30, 2025, https://hiuniversity.com/HIU DOCTORADO MANUAL DE FORMATO DE TESIS DOCTORALES .pdf