INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

DESARROLLO DE SOFTWARE

Instrumento para facilitar el proceso de enseñanza-aprendizaje de la asignatura

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA 20250041

EMPRENDIMIENTO

Período académico
Cuarto

Octubre - 2025

ING. BETTY JARAMILLO TITUAÑA.MEd.

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA – EMPRENDIMEINTO

INSTITUTO SUPERIOR TECNOLÓGICO TENA

Carrera de Tecnología Superior en Desarrollo de Software

ISTT DS Primera Edición – Tena, octubre 2023

SIN ISBN

Instituto Superior Tecnológico Tena
Km. 1 1/2 Vía Tena - Archidona
Tena, Ecuador

AUTORA - REDACCIÓN Y FORMULACIÓN DE CONTENIDOS
Ing. Betty Alexandra Jaramillo Tituaña; MEd.

Profesora del Instituto Superior Tecnológico Tena

REVISIÓN DE PARES

Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango
Mg. Duarte Mora Martha Janina
Lcda. María Campoverde Encalada
Mg. Danilo Alexander Zamora Núñez
Comisión de revisión técnica de guías de estudio del Instituto Superior Tecnológico Tena

APROBACIÓN

Ab. Danilo Alexander Zamora Núñez; Mg.
Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Impreso y hecho en Ecuador.

CONTENIDO

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA

DATOS GENERALES DE LA ASIGNATURA							
Carrera	Desarrollo de Software			**Nombre asignatura**	Emprendimiento		
Modalidad	Presencial			**Campo de Formación**			
Jornada	Matutina y Nocturna			**Unidad de Organización Curricular**	Profesional		
Período académico	Terero			**Código de la asignatura**	DS 401		
Distribución de horas en las actividades de aprendizaje				**N° Total de horas de la asignatura**	48		
N° de horas Docencia	**16**	**N° de horas Aprendizaje Práctico Experimental**				**N° de horas Autónomo**	**8**
		En contacto con docente	**16**	**Autónomo**	**8**		

PRERREQUISITOS Y CORREQUISITOS			
Prerrequisitos de la asignatura		**Correquisitos de la asignatura**	
Asignatura	**Código**	**Asignatura**	**Código**

DESCRIPCIÓN DE LA ASIGNATURA

Emprendimiento es un espacio de pensamiento orientado a la exploración de ideas productivas que motiven al estudiante a la generación de actividades específicas para concretar posibilidades de futuros negocios, sustentables, tiene que ver con el desarrollo de habilidades empresariales y humanas que permitan al emprendedor concretar una idea de negocio en una empresa que brinde nuevos empleos y además permita desarrollar su creatividad y habilidades blandas.

OBJETIVO GENERAL

Desarrollar la capacidad emprendedora y creativa mediante métodos, herramientas y técnicas dinámicas para plantear modelos de negocios innovadores.

CONTRIBUCIÓN DE LOS RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA AL PERFIL DE EGRESO DE LA CARRERA

Resultados de aprendizaje de la asignatura	Resultados de aprendizaje del perfil de egreso de la carrera	Contribución (alta – media – baja)
Aplica habilidades de Tics, trabajo en equipo, gestión de proyectos, liderazgo y creatividad, para trabajar en ambientes colaborativos con profesionalismo y responsabilidad social.	Aplica técnicas de investigación en la búsqueda de nuevas formas de aplicación del desarrollo de software en los sectores industriales.	Alta
Realiza procesos de análisis y verificación de consistencia de datos extraídos de diversas fuentes, que permitan generar reportes relevantes para una organización.	Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación.	Media
Determina los recursos necesarios para el desarrollo de un proyecto software, considerando el hardware, el software y las redes.	Aplica habilidades de Tics, trabajo en equipo, gestión de proyectos, liderazgo y creatividad, para trabajar en ambientes colaborativos con profesionalismo y responsabilidad social.	Media

CONTENIDOS DE LA ASIGNATURA (descripción mínima de contenidos de la asignatura)

Unidad 1 1. Emprendimiento, equidad de género e interculturalidad 1.1 Emprendimientos 1.2 Grupos socioculturales y de género 1.3 Emprendimientos interculturales.
Unidad 2 2. Liderazgo Empresarial. 2.1 Individuos como líderes 2.2 Liderazgo de equipo y organizacional 2.3 Economía circular en la empresa.
Unidad 3 3. Emprendimiento e Innovación. 3.1 Conceptualización de emprendimiento e innovación (design thinking) 3.2 Factores para el emprendimiento 3.3 Tipos de emprendimiento 3.4 Fuentes de innovación para la generación de un emprendimiento
Unidad 4 4. Modelo de Negocio y Marketing Digital 4.1 Generación ideas de negocios 4.2 Estructura del modelo de negocio 4.3 Desarrollo del modelo de negocio 4.4 Fundamentos del plan de marketing digital 4.5 Plataforma de venta digital 4.6 Herramientas de medición, automatización y monitorización en el marketing digital

ESTRATEGIAS METODOLÓGICAS Y RECURSOS DIDÁCTICOS

ESTRATEGIAS METODOLÓGICAS	HABILIDADES BLANDAS	FINALIDAD
Activas para la enseñanza y aprendizaje	**Valores vinculados a la autonomía del sujeto**: confianza, crítica y autocrítica, honestidad, integridad	• Generar confianza/ Promover el pensamiento crítico. • Permite a los estudiantes cumplir un rol activo dentro de su formación. • Construye una sociedad participante.
Aprendizaje y trabajo cooperativo	**Valores elementales de convivencia y civilidad:** crítica y autocrítica, tolerancia, empatía, respeto, justicia, lealtad, paciencia	• Promover un ambiente de colaboración/ trabajo en equipo/ Saber escuchar/Promover el pensamiento crítico/ fomentar el liderazgo/ adaptabilidad. • Mantener una comunicación abierta con el equipo/ tolerancia a los errores, aceptar y aprender de las críticas. • Fomentar el sentido de pertenencia
Aprendizaje individual	**Valores vinculados a la autonomía del sujeto**: responsabilidad, honestidad, integridad, efectividad, autonomía	• Facilitar la asimilación del contenido por parte del estudiante/ Plantear preguntas para promover la comunicación efectiva /Promover el pensamiento crítico • Lectura comprensiva para fijar contenidos/ Promover el pensamiento crítico

RECURSOS DIDÁCTICOS	
MATERIALES CONVENCIONALES	*Material impreso: libros, folletos, fotocopias, periódicos, etc.*
	Tableros didácticos: pizarra
MATERIALES AUDIOVISUALES	*Imágenes fijas proyectables (fotos): diapositivas y fotografías.*
	Materiales audiovisuales (vídeo): películas y vídeos
NUEVAS TECNOLOGÍAS	*Programas informáticos: procesador de palabras, hojas de cálculo, presentaciones*
	Servicios telemáticos: páginas web, plataforma EVA, correo electrónico, google drive

BIBLIOGRAFÍA

Bibliografía Básica de la Asignatura:	**Físico**	**Digital**
Lussier, R y Achua, C. (2016). Liderazgo Teoría, aplicación y desarrollo de habilidades (sexta edición). Cengage Learning S.A. México. ISBN: 978-1-285-8663-2. Número de inventario en biblioteca: ISTT-DS-0047	X	
Muñoz, C. (2020). 100 preguntas clave para todo emprendedor atajos prácticos para ir de cero a emprendedor (primera edición). Penguin Random House Grupo. México. ISBN: 978-607-319-540-9. Número de inventario en biblioteca: ISTT-DS-0135.	X	
Osterwalder, A. y Pigneur, Y. (2010). Generación de modelos de negocio (catorceava edición). Planeta Colombiana S. A. Estados Unidos. ISBN: 978-958- 42-8484-6. Número de inventario en biblioteca: ISTT-ADM-0037	X	
Prieto, C. (2017). Emprendimiento Conceptos y plan de negocios (segunda edición). Pearson. México. ISBN: 978-607-32-4018-5. Número de inventario en biblioteca: ISTT-ADM-0325	X	
Bibliografía de consulta de la Asignatura:	**Físico**	**Digital**
Acosta, J. (2015). Dirigir liderar, motivar, comunicar, delegar, dirigir reuniones (sexta edición). ESIC. España. ISBN: 978-84-7356-905-7. Número de inventario en biblioteca: ISTT-ADM-0025	X	
Barg , G. y Pietersma, P. (2016) Modelos clave de negocios Más de 75 modelos que todo gerente necesita saber (primera edición. Miembro de cámara nacional de la industria. México. ISBN: 978-607-17-2770-1. Número de inventario en biblioteca: ISTT-ADM-0017	X	

DESCRIPCIÓN DE LA ASIGNATURA

La asignatura de Emprendimiento e Innovación está diseñada para desarrollar en los estudiantes capacidades emprendedoras, creativas e innovadoras que les permitan identificar oportunidades de negocio en el sector tecnológico y de desarrollo de software. Se enfoca en la comprensión de conceptos fundamentales del emprendimiento TECH, el desarrollo de habilidades de liderazgo técnico y de equipos de desarrollo, la generación de ideas innovadoras de productos digitales y la creación de modelos de negocio sostenibles en la industria del software.

La asignatura integra perspectivas de equidad de género e interculturalidad, reconociendo la diversidad como fuente de innovación en equipos de desarrollo y en la creación de soluciones tecnológicas inclusivas y accesibles. Los estudiantes aprenderán a aplicar metodologías como Design Thinking y metodologías ágiles, a desarrollar modelos de negocio utilizando herramientas como el Business Model Canvas y Lean Startup, y a implementar estrategias de marketing digital específicas para productos de software, y estrategias de posicionamiento en plataformas tecnológicas.

Esta materia constituye una herramienta fundamental para que los futuros profesionales en Desarrollo de Software puedan crear, gestionar y desarrollar proyectos emprendedores tecnológicos que contribuyan al desarrollo económico y social de sus comunidades, aprovechando las oportunidades que ofrece la transformación digital en Ecuador y la creciente demanda de soluciones de software innovadoras.

OBJETIVO GENERAL

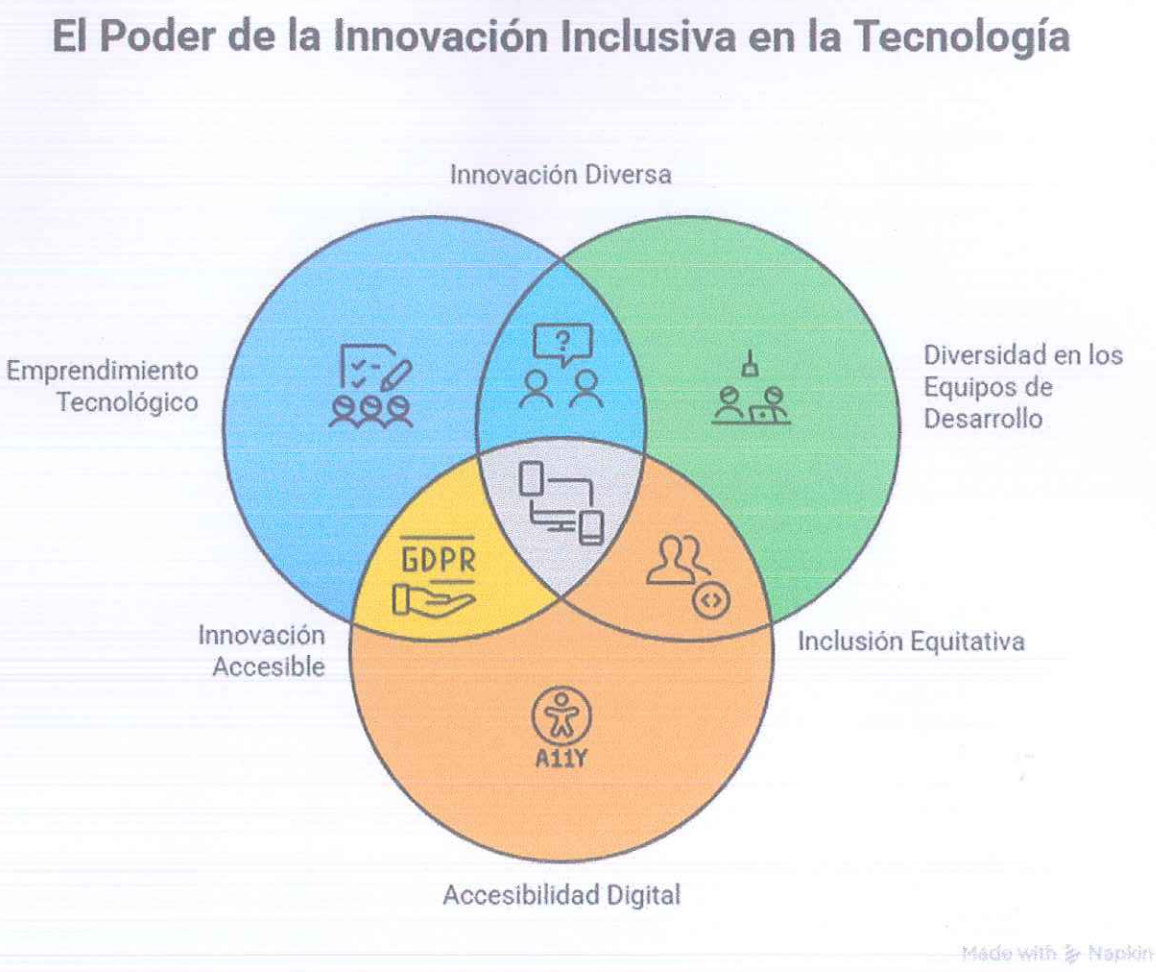
Desarrollar competencias emprendedoras e innovadoras en los estudiantes mediante el análisis de oportunidades de negocio tecnológico, la aplicación de metodologías de innovación y desarrollo ágil, y la creación de modelos de negocio sostenibles en la industria del software, incorporando principios de equidad de género, interculturalidad, accesibilidad digital y desarrollo de software sostenible, para contribuir a la transformación digital y al desarrollo económico y social del Ecuador.

DESCRIPTIVA DE LAS COMPETENCIAS DE LA GUÍA DE EMPRENDIMIENTO E INNOVACIÓN

Este documento presenta una descripción detallada de las competencias que se desarrollan en la asignatura de Emprendimiento e Innovación. En un contexto globalizado de transformación digital acelerada y constante evolución tecnológica, es fundamental que los estudiantes de Desarrollo de Software adquieran habilidades que les permitan identificar oportunidades, crear valor mediante soluciones tecnológicas innovadoras y desarrollar proyectos de software sostenibles y escalables que respondan a necesidades reales del mercado ecuatoriano e internacional. La guía está diseñada para proporcionar a los estudiantes una comprensión integral del ecosistema emprendedor tecnológico, desde la concepción de ideas innovadoras de productos digitales hasta la implementación de modelos de negocio viables en la industria del software. Se enfatiza el desarrollo de competencias blandas como el liderazgo técnico, la creatividad aplicada a la resolución de problemas mediante código, el trabajo en equipo multidisciplinario (con diseñadores, product managers, stakeholders), la comunicación efectiva de conceptos técnicos, y la adaptabilidad ante cambios tecnológicos rápidos, junto con habilidades técnico-empresariales relacionadas con la arquitectura de software, gestión de productos digitales, metodologías ágiles, analítica de datos, y marketing digital específico para productos tech.

UNIDAD 1: EMPRENDIMIENTO, EQUIDAD DE GÉNERO E INTERCULTURALIDAD

DIAGRAMA DE APRENDIZAJE



SÍNTESIS

Esta unidad introduce los conceptos fundamentales del emprendimiento tecnológico desde una perspectiva inclusiva e intercultural. Se explora cómo la diversidad en equipos de desarrollo y la consideración de diferentes contextos socioculturales constituyen elementos clave para la innovación y el desarrollo de soluciones de software sostenibles y accesibles. Los estudiantes comprenderán que el emprendimiento en tecnología no solo es una actividad económica, sino también un motor de transformación social que puede contribuir a reducir brechas digitales y promover el desarrollo equitativo de comunidades diversas.

Resultado de Aprendizaje: El estudiante será capaz de identificar oportunidades de emprendimiento tecnológico considerando la equidad de género y la interculturalidad en el

desarrollo de software, desarrollando propuestas inclusivas que valoren la diversidad como fuente de innovación y que generen soluciones tecnológicas accesibles para diferentes contextos socioculturales.

1.1 EMPRENDIMIENTOS EN TECNOLOGÍA

Concepto de Emprendimiento Tecnológico

El emprendimiento tecnológico es el proceso mediante el cual individuos o equipos identifican oportunidades, desarrollan soluciones de software innovadoras y crean valor a través de la tecnología. Según Prieto (2017), el emprendimiento implica la "identificación de oportunidades y la movilización de recursos para transformarlas en proyectos viables" (p. 23). En el contexto tecnológico, esto se traduce en transformar problemas en soluciones digitales que mejoren procesos, faciliten acceso a servicios o creen nuevas experiencias para usuarios.

El emprendimiento en desarrollo de software va más allá de la simple creación de empresas tecnológicas. Implica:

- **Identificación de problemas tecnológicos:** La capacidad de reconocer ineficiencias, procesos manuales susceptibles de automatización, o necesidades no satisfechas que pueden resolverse mediante software.
- **Desarrollo de soluciones escalables:** La habilidad para crear productos digitales que puedan crecer sin incrementos proporcionales de costos, aprovechando la naturaleza del software como activo reproducible.
- **Creación de valor digital:** El proceso de generar beneficios económicos, sociales o de eficiencia a través de aplicaciones, plataformas, sistemas o servicios digitales.
- **Innovación tecnológica continua:** La disposición a experimentar con nuevas tecnologías, frameworks, arquitecturas y paradigmas de programación para mantener relevancia competitiva.

Características del Emprendedor en Desarrollo de Software

Según Muñoz (2020), el perfil emprendedor incluye características personales y competencias específicas. En el contexto del desarrollo de software, estas se manifiestan de manera particular:

- **Pensamiento computacional:** Capacidad de descomponer problemas complejos en componentes manejables, identificar patrones, abstraer conceptos y diseñar algoritmos para resolverlos sistemáticamente.
- **Visión producto-tecnológica:** Habilidad para imaginar soluciones tecnológicas futuras y establecer roadmaps que equilibren viabilidad técnica con valor de usuario, priorizando features según impacto.
- **Aprendizaje continuo:** La industria tecnológica evoluciona rápidamente. Los emprendedores tech exitosos mantienen curiosidad constante, aprenden nuevos lenguajes, frameworks y paradigmas, adaptándose a cambios del ecosistema.
- **Resiliencia ante bugs y fallos:** En desarrollo de software, el fracaso es iterativo: bugs, features que no funcionan como se esperaba, pivotes técnicos. La capacidad de depurar, aprender de errores y persistir es fundamental.
- **Orientación a datos:** Tomar decisiones basadas en métricas de producto (engagement, retención, conversión) y datos de usuarios reales, no solo intuición o preferencias personales.
- **Colaboración técnica:** Capacidad de trabajar en equipos multidisciplinarios, realizar code reviews constructivos, documentar código para otros, y comunicar conceptos técnicos a stakeholders no técnicos.
- **Mentalidad de producto:** Pensar más allá del código, considerando experiencia de usuario, modelo de negocio, go-to-market, soporte y evolución del producto a largo plazo.

Tipos de Emprendimiento Tecnológico

Basándose en Prieto (2017) y adaptado al contexto del software, podemos clasificar:

Según motivación:

- **Emprendimiento por oportunidad tecnológica:** Surge de identificar una oportunidad de mercado o necesidad no satisfecha que puede resolverse mediante software innovador.
  Ejemplo: Identificar que las PYMEs ecuatorianas carecen de sistemas de facturación electrónica accesibles y desarrollar un SaaS específico.
- **Emprendimiento por necesidad:** Nace como respuesta a falta de alternativas laborales o búsqueda de independencia económica mediante desarrollo freelance o productos propios.
  Ejemplo: Desarrollador que crea plugins de WordPress para generar ingresos pasivos tras perder empleo formal.

Según impacto:

- **Emprendimiento de subsistencia digital:** Genera ingresos básicos mediante desarrollo freelance, pequeños proyectos o mantenimiento de sistemas.
- **Emprendimiento de crecimiento (Growth startups):** Busca escalar rápidamente aprovechando la naturaleza escalable del software, con ambición de multiplicar usuarios y valor.

  Ejemplo: Startup de e-commerce con arquitectura cloud que puede atender 100 o 100,000 tr ansacciones con infraestructura similar.
- **Emprendimiento transformador (Disruptive TECH):** Pretende cambiar industrias completas mediante innovación tecnológica radical.

  Ejemplo: Uber transformando transporte, Airbnb transformando hotelería mediante plataformas digitales.

Según propósito:

- **Emprendimiento comercial TECH:** Orientado principalmente a generar beneficios económicos mediante productos de software exitosos.
- **Emprendimiento social tech (Tech for Good):** Busca resolver problemas sociales o ambientales mientras genera sostenibilidad financiera mediante tecnología.

  Ejemplo: Plataforma que conecta donantes con escuelas rurales para proveer equipos tecnológicos y capacitación digital.
- **Emprendimiento open source:** Centrado en desarrollar software libre mientras se monetiza mediante servicios, soporte, versiones enterprise o modelos freemium.

  Ejemplo: Red Hat, WordPress, MongoDB que ofrecen software libre con servicios comerciales.

Ecosistema Emprendedor Tecnológico

Según Muñoz (2020), el ecosistema emprendedor incluye actores, instituciones y recursos que facilitan la actividad emprendedora. En tecnología, este ecosistema tiene características particulares:

Componentes del ecosistema TECH en Ecuador:

Incubadoras y aceleradoras tecnológicas:

- Kruger Labs (Quito)
- Prendho (Cuenca)
- ImpactHub
- Programas universitarios (EPN, ESPOL)
- Wayra (Telefónica)

Espacios de coworking TECH:

- Impaqto (Quito)
- Co+Labo
- Espacios maker con equipamiento tecnológico

Comunidades y eventos TECH:

- Meetups de desarrollo (JavaScript Ecuador, Python Ecuador)
- Hackathons (NASA Space Apps, AngelHack)
- Conferencias (JSConf, DevFest)
- Comunidades online (Discord, Slack groups)

Acceso a educación tecnológica:

- Bootcamps (Laboratoria, Holberton School)
- Plataformas online (Platzi, Udemy, Coursera)
- Universidades con carreras de software
- Recursos gratuitos (freeCodeCamp, The Odin Project)

Financiamiento TECH:

- Inversores ángeles tech-focused
- Fondos de capital de riesgo (VC) para startups tecnológicas
- Crowdfunding en plataformas como Kickstarter (para productos)
- Grants de empresas tech (Google, Microsoft)
- Concursos de innovación tecnológica

Infraestructura tecnológica:

- Acceso a internet de alta velocidad
- Cloud computing (AWS, Google Cloud, Azure)
- Herramientas de desarrollo gratuitas o accesibles
- APIs y servicios de terceros
- GitHub, GitLab para colaboración de código

Regulación y políticas:

- Ley Orgánica de Emprendimiento e Innovación (2020)
- Políticas de protección de datos
- Regulaciones específicas por sector (fintech, healthtech)
- Propiedad intelectual de software

Emprendimiento en el Sector del Desarrollo de Software

El desarrollo de software ofrece ventajas únicas como sector emprendedor:

Barreras de entrada bajas:

- No requiere capital inicial significativo (comparado con manufactura)
- Laptop + internet + conocimiento = capacidad de crear productos
- Herramientas de desarrollo mayormente gratuitas (IDEs, frameworks open source)
- Infraestructura cloud con pricing de pago por uso

Escalabilidad inherente:

- Costo marginal cercano a cero por usuario adicional
- Distribuir software no requiere producción física adicional
- Actualizaciones globales instantáneas
- Potencial de alcance mundial desde día uno

Áreas de oportunidad en Ecuador:

Software empresarial (B2B SaaS):

- ERP para PYMEs ecuatorianas
- CRM adaptado a contexto local
- Software de facturación electrónica
- Sistemas de gestión para sectores específicos (agrícola, retail, educación)

Fintech:

- Pasarelas de pago locales
- Soluciones de préstamos P2P
- Billeteras digitales
- Herramientas de educación financiera

Edtech:

- Plataformas de aprendizaje online
- Sistemas de gestión educativa
- Gamificación educativa
- Herramientas para educación rural

Healthtech:

- Telemedicina
- Gestión de historias clínicas
- Apps de salud y bienestar
- Sistemas de turnos y gestión hospitalaria

Agritech:

- Software de agricultura de precisión
- Marketplaces agrícolas

- Sistemas de trazabilidad
- Monitoreo mediante IoT y sensores

Govtech:

- Soluciones de gobierno electrónico
- Participación ciudadana digital
- Transparencia y datos abiertos
- Servicios públicos digitalizados

E-commerce y marketplaces:

- Plataformas de comercio electrónico especializadas
- Marketplaces verticales por industria
- Logística de última milla optimizada por software

Barreras al Emprendimiento Tecnológico

Según Prieto (2017), los emprendedores enfrentan obstáculos diversos. En el contexto tecnológico ecuatoriano:

Acceso limitado a capital de riesgo:

- Ecosistema de VC aún en desarrollo
- Preferencia de inversores por startups en etapas más avanzadas
- Desconfianza en emprendedores sin track record
- Montos de inversión menores comparados con otros países de la región

Fuga de talento (brain drain):

- Desarrolladores talentosos emigran buscando mejores oportunidades
- Salarios internacionales remotos compiten con emprendimientos locales
- Dificultad para retener talento técnico senior

Infraestructura digital desigual:

- Conectividad limitada en zonas rurales
- Brecha digital limita mercado potencial
- Costos de internet relativamente altos

Marco regulatorio en evolución:

- Normativas para fintech, healthtech aún desarrollándose
- Incertidumbre regulatoria en nuevas tecnologías
- Complejidad para cumplir regulaciones internacionales (GDPR)

Mercado local pequeño:

- Población ecuatoriana de ~18 millones limita escalabilidad

- Necesidad de pensar regionalmente desde inicio
- Preferencia por software internacional "probado"

Educación tecnológica con gaps:

- Curricula universitaria a veces desactualizada
- Falta de formación en habilidades prácticas (soft skills, negocio)
- Escasez de mentores con experiencia en emprendimiento tech exitoso

Cultura de aversión al riesgo:

- Presión social por empleo "estable"
- Estigma del fracaso empresarial
- Preferencia por carreras corporativas tradicionales

1.2 GRUPOS SOCIOCULTURALES Y DE GÉNERO EN TECNOLOGÍA

Diversidad y Emprendimiento Tecnológico

Lussier y Achua (2016) destacan que "la diversidad en equipos, cuando se gestiona efectivamente, resulta en mayor innovación y mejores decisiones" (p. 387). En desarrollo de software, esto es particularmente relevante dado que los productos tecnológicos impactan a audiencias diversas globalmente.

La diversidad en emprendimiento tecnológico aporta:

Perspectivas múltiples en diseño de producto:

- Diferentes experiencias de vida identifican problemas diversos
- Variedad de enfoques para resolver un mismo desafío técnico
- Cuestionamiento de supuestos sobre "usuarios típicos"
- Detección de sesgos en algoritmos y sistemas

Innovación en soluciones técnicas:

- Combinación de diferentes backgrounds técnicos (frontend, backend, móvil, datos)
- Fusión de conocimientos de diferentes industrias aplicados a tech
- Creatividad potenciada por diversidad cognitiva

Acceso a mercados diversos:

- Mejor comprensión de necesidades de diferentes segmentos
- Capacidad de diseñar para contextos culturales variados
- Productos más inclusivos y accesibles por diseño

Legitimidad y adopción:

- Equipos diversos generan mayor confianza en comunidades diversas
- Representación importa para adoption de productos

- Mejores prácticas de localización e internacionalización

Equidad de Género en Tecnología

A pesar de los avances, persisten brechas de género significativas en la industria tecnológica a nivel global y en Ecuador:

Situación actual:

- Las mujeres representan aproximadamente 20-30% de profesionales en tecnología en Ecuador
- Porcentaje menor en roles de liderazgo técnico y founders de startups
- Subrepresentación mayor en áreas como ciberseguridad, AI/ML, infraestructura
- Mayor presencia en UX/UI, frontend, QA (aunque esto también refleja estereotipos)

Barreras específicas que enfrentan las mujeres en TECH:

Estereotipos de género en STEM:

- Socialización que desalienta interés temprano en tecnología
- Mensajes culturales: "programación es para hombres"
- Falta de modelos de rol femeninos visibles en tech
- Síndrome del impostor más pronunciado

Ambiente de trabajo poco inclusivo:

- "Bro culture" en muchas startups y empresas tech
- Microagresiones y mansplaining en reuniones técnicas
- Exclusión de redes informales donde se toman decisiones
- Falta de políticas de conciliación laboral-familiar

Acceso limitado a financiamiento:

- Fundadoras reciben ~2% del capital de riesgo globalmente
- Sesgos inconscientes de inversionistas (mayoría hombres)
- Menores montos de inversión cuando lo obtienen
- Preguntas diferentes en pitch: a hombres sobre oportunidades, a mujeres sobre riesgos

Brecha salarial en tech:

- Diferencias salariales para roles equivalentes
- Menor compensación en equity/stock options
- Negociaciones salariales afectadas por sesgos

Educación y pipeline:

- Menor matrícula femenina en carreras de CS e ingeniería de software
- Deserción más alta de mujeres en carreras tech

- Menos acceso a bootcamps y programas intensivos por carga de cuidado

Estrategias para promover equidad de género en emprendimiento TECH:

Educación temprana:

- Programas como Girls Who Code en escuelas
- Talleres de programación para niñas
- Comunicación que desmitifica la tecnología como "solo para hombres"
- Mentorías desde edades tempranas

Comunidades de mujeres en tech:

- Women Who Code Ecuador
- PyLadies
- Django Girls
- TechWomen
- Redes de networking y soporte mutuo

Políticas organizacionales:

- Hiring diverso intencionalmente (blind resume reviews)
- Trabajo remoto y horarios flexibles
- Licencias parentales equitativas
- Protocolos claros contra acoso y discriminación

Programas de aceleración específicos:

- Aceleradoras para fundadoras tech
- Fondos de inversión con perspectiva de género
- Mentorías con empresarias tech exitosas

Visibilización:

- Conferencias con speakers mujeres
- Publicaciones sobre fundadoras exitosas
- Premios y reconocimientos a mujeres en tech
- Media highlighting female role models

Diversidad Cultural e Intercultural en Desarrollo de Software

Ecuador es un país plurinacional y multicultural. Esta diversidad presenta oportunidades únicas para emprendimiento tecnológico inclusivo:

Grupos socioculturales en Ecuador:

Pueblos indígenas:

- 14 nacionalidades y 18 pueblos
- Kichwa (Sierra y Amazonía), Shuar, Achuar, Waorani, Tsáchila, Awá, etc.

- Lenguas originarias diversas
- Conocimientos ancestrales valiosos

Pueblo afroecuatoriano:

- Principalmente en Esmeraldas, Valle del Chota, Guayaquil
- Historia y cultura específicas
- Tradiciones orales ricas

Pueblo montubio:

- Zonas rurales costeras
- Identidad cultural propia

Mestizos y población urbana diversa:

- Diferentes contextos socioeconómicos
- Diversidad regional (Costa, Sierra, Amazonía, Galápagos)

Brechas digitales que afectan a grupos diversos:

Acceso a tecnología:

- 54.1% de hogares ecuatorianos con internet (INEC, 2020)
- Brecha urbano-rural significativa
- Menor acceso en comunidades indígenas y rurales
- Equipamiento tecnológico limitado

Alfabetización digital:

- Diferentes niveles de competencias digitales
- Barrera idiomática (contenido mayormente en español/inglés)
- Falta de capacitación contextualizada

Contenido culturalmente relevante:

- Software y aplicaciones diseñadas desde perspectivas urbanas occidentales
- Falta de localización a lenguas originarias
- Interfaces que no consideran diversidad cultural

Oportunidades del emprendimiento tecnológico intercultural:

Localización e internacionalización:

- Desarrollar software en lenguas originarias (Kichwa, Shuar, etc.)
- Interfaces adaptadas culturalmente
- Iconografía y diseño contextual
- Ejemplo: App educativa para aprender Kichwa mediante gamificación

Soluciones para contextos diversos:

- Apps funcionando offline (zonas sin conectividad constante)

- SMS-based solutions (sin requerir smartphones)
- Diseño para low-end devices
- Ejemplo: Sistema de alertas agrícolas via SMS para comunidades rurales

Preservación de conocimiento ancestral:

- Plataformas de documentación de lenguas originarias
- Bases de datos de medicina tradicional
- Archivo digital de tradiciones orales
- **Ejemplo:** App de realidad aumentada que narra leyendas indígenas en ubicaciones específicas

Inclusión financiera digital:

- Soluciones fintech para poblaciones no bancarizadas
- Interfaces simplificadas para población con baja alfabetización
- Ejemplo: Billetera digital con interfaz visual para comerciantes rurales

Educación intercultural:

- Plataformas educativas con contenido en lenguas originarias
- Metodologías pedagógicas adaptadas culturalmente
- Ejemplo: Sistema de gestión educativa para escuelas interculturales bilingües

Conexión mercados:

- Marketplaces que conectan productores rurales/indígenas con compradores
- Sistemas de trazabilidad para productos artesanales
- Ejemplo: Plataforma que certifica autenticidad de artesanías y comparte historia del artesano

Desafíos del emprendimiento tecnológico intercultural:

Diseño participativo:

- Necesidad de co-crear con comunidades, no "para" ellas
- Investigación de usuarios profunda y respetuosa
- Evitar extractivismo de conocimiento

Apropiación cultural y propiedad intelectual:

- Proteger conocimientos tradicionales
- Evitar apropiación de símbolos y narrativas
- Compensación justa y reconocimiento

Sostenibilidad comunitaria:

- Soluciones que empoderan, no generan dependencia
- Capacitación para mantenimiento local

- Apropiación tecnológica por las comunidades

Conectividad y electricidad:

- Infraestructura básica limitada en zonas rurales
- Diseño offline-first
- Optimización de batería y datos

Principios para Emprendimientos Tecnológicos Inclusivos

Basándose en Lussier y Achua (2016) sobre liderazgo inclusivo, aplicado a emprendimiento tech:

Diseño universal y accesibilidad:

- WCAG (Web Content Accessibility Guidelines)
- Diseño para personas con discapacidades desde el inicio
- Teclado navigation, screen readers, contraste de colores
- Closed captions, transcripciones
- Testing con usuarios diversos

Lenguaje inclusivo en UI/UX:

- Opciones de género no binarias
- Nombres que acomodan diferentes culturas
- Fechas en formatos diversos
- Sensibilidad cultural en copy

Datos y algoritmos sin sesgos:

- Auditar datasets por representatividad
- Testing de modelos ML por subgrupos demográficos
- Documentar limitaciones y sesgos potenciales
- Equipos diversos revisando algoritmos

Investigación de usuarios diversa:

- Reclutar participants de diversos backgrounds
- Compensar justamente tiempo de usuarios
- Investigación en contextos reales diversos
- Considerar diferentes niveles de alfabetización digital

Equipos de desarrollo diversos:

- Hiring intencional de perfiles diversos
- Crear ambientes psicológicamente seguros
- Valorar diferentes perspectivas en decisiones técnicas
- Mentoría y growth para grupos subrepresentados

1.3 EMPRENDIMIENTOS INTERCULTURALES EN TECNOLOGÍA

Modelos de Emprendimiento Tecnológico Intercultural

Basándose en Prieto (2017) sobre modelos de negocio, aplicado a emprendimientos tech interculturales:

Desarrollo de Software Participativo (Community-Based Tech)

Modelo donde comunidades co-crean soluciones tecnológicas para sus propias necesidades.

Características:

- Toma de decisiones participativa sobre features y roadmap
- Desarrollo iterativo con feedback constante de la comunidad
- Capacitación técnica a miembros de la comunidad
- Código open source cuando es posible
- Ownership compartido del producto

Proceso:

1. Diagnóstico participativo: Talleres con comunidad para identificar necesidades tecnológicas reales
2. Co-diseño: Sesiones de design thinking conjuntas, wireframing colaborativo
3. Desarrollo iterativo: Sprints cortos con demos frecuentes, ajustes basados en feedback
4. Capacitación: Train-the-trainer para soporte local, documentación accesible
5. Mantenimiento comunitario: Transferencia gradual de capacidades técnicas

Ejemplo: Sistema de gestión de recursos naturales para comunidad indígena amazónica:

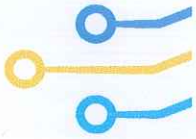
- Co-diseñado con líderes comunitarios y guardaparques locales
- Interfaz en español y Kichwa
- Funciona offline con sincronización posterior
- Mapeo de biodiversidad y alertas de deforestación
- Capacitación a jóvenes de la comunidad en mantenimiento

Ventajas:

- Alta adopción por ownership comunitario
- Solución realmente ajustada a necesidades
- Empoderamiento tecnológico local
- Sostenibilidad a largo plazo

Desafíos:

- Proceso más lento que desarrollo tradicional



- Requiere habilidades de facilitación, no solo técnicas
- Balance entre visión técnica y deseos diversos de la comunidad
- Gestión de expectativas sobre tiempos

Cooperativas Tecnológicas

Desarrolladores de diversos backgrounds se organizan cooperativamente.

Características:

- Estructura democrática (un socio, un voto)
- Distribución equitativa de beneficios
- Proyectos con impacto social
- Capacitación cruzada entre miembros

Ejemplo: Cooperativa de desarrollo de software que:

- Miembros de diferentes nacionalidades y backgrounds
- Prioriza proyectos para ONGs, gobiernos locales, emprendimientos sociales
- Utiliza excedentes para becas tech para comunidades marginadas
- Rotación de roles técnicos para desarrollo de todos

Startups con Impacto Social Diverso

Startups tech que integran impacto social en su core business model.

Características:

- Modelo de negocio viable que también genera impacto
- Métricas duales: financieras y de impacto social
- Compromiso con diversidad en equipo y beneficiarios

Modelos de impacto:

- One-for-one digital: Por cada licencia vendida, una gratuita para escuela rural
- Freemium social: Versión gratuita robusta para ONGs y organizaciones sociales
- Graduated pricing: Precio según capacidad de pago (empresas pagan más, PYMEs menos)
- Porcentaje de ingresos: X% de revenue para programas sociales

Ejemplo: Plataforma de telemedicina:

- Conecta pacientes rurales con médicos urbanos
- Empresas pagan suscripción, comunidades rurales acceso subsidiado
- Equipo incluye médicos indígenas y occidentales
- Interfaz adaptada a baja alfabetización digital
- 10% de ingresos a capacitación de promotores de salud comunitarios

Alianzas Estratégicas Interculturales

Colaboraciones entre startups tech, comunidades, ONGs, gobierno y academia.

Tipos de alianzas:

- Tech + Comunidad: Startup provee tecnología, comunidad provee contexto y testing
- Tech + ONG: Startup provee plataforma, ONG provee reach y confianza comunitaria
- Tech + Academia: Startup comercializa investigación, universidad provee I+D
- Tech + Gobierno: Startup provee solución, gobierno escala y financia
- Multistakeholder: Combinación de varios actores

Ejemplo: Plataforma educativa digital:

- Startup tech: desarrollo y mantenimiento
- Ministerio de Educación: contenido curricular y financiamiento
- Universidad: investigación pedagógica y evaluación de impacto
- Comunidades: piloto y feedback
- ONG: capacitación docente
- ISPs: conectividad subsidiada

Diseño de Emprendimientos Tecnológicos Interculturales

User Research Intercultural

Metodologías de investigación adaptadas a contextos diversos:

Preparación:

- Investigar contexto cultural previamente
- Identificar gatekeepers y autoridades comunitarias
- Obtener permisos apropiados
- Considerar tiempos comunitarios (no apresurarse)
- Compensación justa por tiempo de participantes

Métodos adaptados:

Entrevistas contextuales:

- En lugares donde usuarios están cómodos (no oficinas)
- En lengua preferida del usuario (intérpretes si es necesario)
- Preguntas abiertas, narrativas
- Observación de uso de tecnología actual

Talleres participativos:

- Card sorting con imágenes (no solo texto)
- Storyboarding visual
- Role-playing de escenarios de uso

- Co-diseño de interfaces con materiales físicos

Diarios de usuario:

- Captura de experiencias cotidianas relevantes
- Puede ser mediante fotos (no solo texto)
- Seguimiento por WhatsApp (herramienta familiar)

Contextual inquiry:

- Observar en contexto real de uso
- Shadowing respetuoso
- Preguntar "muéstrame cómo lo haces"

Consideraciones éticas:

- Consentimiento informado claro
- Privacidad y confidencialidad
- Devolución de resultados a la comunidad
- No extractivismo de conocimiento

Diseño de Interfaces Interculturales

Principios para UI/UX inclusiva culturalmente:

Localización profunda (no solo traducción):

- Lengua nativa en UI
- Unidades de medida locales (hectáreas, libras, etc.)
- Formatos de fecha culturalmente apropiados
- Nombres y apellidos con estructura flexible
- Moneda local

Iconografía culturalmente relevante:

- Símbolos que resuenen localmente
- Evitar iconos culturalmente específicos de occidente
- Testing con usuarios target de comprensión
- Ilustraciones con representación diversa

Arquitectura de información adaptada:

- Jerarquía que refleje prioridades culturales
- Categorización basada en modelos mentales locales
- Navegación que considere diferentes niveles de alfabetización digital

Diseño para contextos de uso reales:

- Usabilidad con guantes (agricultura)
- Legibilidad bajo sol (outdoor)

- Diseño para uso compartido de dispositivos
- Optimización para low-end devices

Accesibilidad inclusiva:

- Soporte para screen readers en lengua local
- Alto contraste para uso en diferentes condiciones de luz
- Text size ajustable
- Navegación clara para usuarios novatos

Desarrollo Técnico Inclusivo

Stack tecnológico considerando contexto:

- Progressive Web Apps: Funcionan en navegadores, no requieren app store
- Offline-first: Sincronización cuando hay conexión
- SMS APIs: Alertas sin requerir smartphone
- USSD codes: Interacción sin internet
- Lightweight frameworks: Funcionan en dispositivos de bajos recursos

Performance optimization para dispositivos de bajos recursos:

- Lazy loading de imágenes y componentes
- Compresión de assets
- Minificación de código
- CDN para recursos estáticos
- Caching estratégico

Consideraciones de datos:

- Minimizar uso de datos móviles
- Comprimir requests/responses
- Permitir descargas offline de contenido esencial

Capacitación y Transferencia Tecnológica

Para sostenibilidad a largo plazo:

Train-the-trainer:

- Identificar champions tecnológicos en comunidad
- Capacitación profunda a grupo pequeño
- Ellos capacitan a otros (efecto multiplicador)
- Materiales de capacitación en lengua local

Documentación accesible:

- Videos tutoriales (no solo texto)
- Screenshots con anotaciones claras

- FAQs basadas en preguntas reales
- Hotline o WhatsApp de soporte

Soporte comunitario:

- Grupos de WhatsApp o Telegram para troubleshooting peer-to-peer
- Moderadores capacitados de la comunidad
- Escalación a equipo técnico solo para issues complejos

Empoderamiento técnico:

- Enseñar no solo uso, también conceptos básicos de cómo funciona
- Talleres de programación básica para interesados
- Oportunidades de employment en el proyecto para miembros comunitarios

Casos de Estudio: Emprendimientos Tecnológicos Interculturales

Caso 1: App Educativa Multilingüe "Yachay" (Ecuador)

Contexto: Startup ecuatoriana que desarrolla aplicación educativa para niños de comunidades bilingües español-kichwa.

Modelo de negocio:

- Freemium: Contenido básico gratuito, premium con más actividades
- Licencias para escuelas (B2B)
- Donaciones de organizaciones internacionales

Desarrollo intercultural:

- Equipo incluye hablantes nativos de Kichwa
- Contenido co-creado con maestros de escuelas interculturales bilingües
- Interfaz en español, Kichwa unificado, y Kichwa con variantes dialectales
- Ilustraciones con representación de niños indígenas
- Historias basadas en cosmovisión andina

Factores de éxito:

- Consulta constante con comunidades educativas
- Respeto por variantes lingüísticas (Kichwa no es monolítico)
- Diseño atractivo que compite con apps internacionales
- Funciona offline (muchas escuelas rurales sin internet constante)
- Prueba piloto extensa antes de lanzamiento comercial

Desafíos superados:

- Estandarización de Kichwa escrito (múltiples variantes)
- Financiamiento inicial (grants de fundaciones)
- Balance entre visión pedagógica occidental y andina

- Retención de usuarios en contextos donde los dispositivos son compartidos

Impacto:

- 15,000 estudiantes usuarios activos
- Presencia en 50 escuelas rurales
- Mejora medible en habilidades de lectura en ambas lenguas
- Orgullo cultural incrementado (según testimonios docentes)

Lecciones:

- La tecnología puede preservar lenguas amenazadas
- Co-creación no es solo consultar, es diseñar juntos
- Modelo de negocio debe ser híbrido (comercial + social)
- Paciencia: el proceso intercultural toma tiempo

Caso 2: Plataforma "ConectAgro" - Marketplace Agrícola

Contexto: Plataforma que conecta pequeños agricultores rurales (muchos indígenas) con compradores urbanos, eliminando intermediarios.

Modelo de negocio:

- Comisión por transacción (5% al comprador, 0% al agricultor inicialmente)
- Suscripción premium para compradores frecuentes (restaurantes, hoteles)
- Servicios adicionales: logística, certificaciones

Desarrollo intercultural:

- Interfaces diferenciadas: App simplificada para agricultores, web completa para compradores
- Soporte SMS para agricultores sin smartphones
- Registro mediante WhatsApp con asistencia humana
- Perfil del agricultor incluye su historia, fotos de su parcela
- Sistema de calificación bidireccional

Stack técnico:

- PWA (Progressive Web App) para agricultores
- Backend Node.js + PostgreSQL
- Integración con Twilio para SMS
- Maps integration para logística
- Sistema de pagos con opción de efectivo en entrega

Factores de éxito:

- Propuesta de valor clara para ambos lados
- Generación de confianza mediante perfiles detallados

- Soporte multicanal (app, SMS, llamadas)
- Capacitación presencial a agricultores en ferias locales
- Partnership con cooperativas agrícolas para onboarding

Impacto:

- 500 agricultores activos
- Incremento promedio de 30% en ingresos de agricultores
- Reducción de desperdicios (conectan directamente)
- Compradores acceden a productos más frescos
- Trazabilidad: compradores saben de dónde viene su comida

Desafíos:

- Logística de última milla en zonas rurales
- Conectividad intermitente requiere diseño offline-first robusto
- Diferentes niveles de alfabetización digital
- Construcción de confianza inicial (reticencia a plataformas)
- Estacionalidad de productos

Escalabilidad:

- Modelo replicable en otras regiones
- Potencial de expansión a otros productos (artesanías, lácteos)
- Datos generados valiosos para análisis de mercado

Caso 3: Sistema de Salud Comunitaria "Jambi" (Salud en Kichwa)

Contexto: Plataforma de telemedicina y gestión de salud comunitaria para zonas rurales amazónicas.

Modelo de negocio:

- B2B2C: Ministerio de Salud paga licencias, comunidades acceden gratuitamente
- Fundaciones internacionales financian extensión
- Alianza con operadoras telefónicas para data patrocinada (zero-rating para la app)

Características interculturales:

- Médicos indígenas y occidentales en plataforma
- Respeto por medicina ancestral: app incluye remedios tradicionales junto a occidentales
- Interfaz gráfica minimalista (íconos claros)
- Consultas por videollamada, voz, o mensajes
- Traducción automática español-kichwa en tiempo real (IA entrenada)

Tecnología:

- Flutter para app móvil multiplataforma
- WebRTC para videollamadas de baja latencia
- ML para traducción español-kichwa
- Compresión de video para funcionar con baja conectividad
- Sincronización de registros médicos offline

Componentes del sistema:

1. App paciente: Solicitar consultas, ver historial, recordatorios de medicación
2. App promotor de salud: Registro de casos, alertas epidemiológicas
3. Dashboard médico: Gestión de consultas, historial de pacientes
4. Panel analytics: Ministerio de Salud visualiza datos epidemiológicos en tiempo real

Factores de éxito:

- Alianza gobierno-startup-comunidades desde inicio
- Piloto extenso con retroalimentación continua
- Capacitación a promotores de salud comunitarios
- Inclusión de medicina tradicional (no solo occidental)
- Respuesta rápida durante pandemia COVID-19 incrementó adoption

Impacto:

- 30 comunidades amazónicas conectadas
- Reducción de 60% en viajes innecesarios a centros de salud urbanos
- Detección temprana de brotes epidémicos
- Preservación de conocimiento de medicina ancestral (documentado)
- Mejora en adherencia a tratamientos (recordatorios automáticos)

Lecciones clave:

- Alianzas público-privadas son cruciales para escala
- Respeto cultural (medicina tradicional) es fundamental para adopción
- Tecnología debe ser invisible: usuarios no piensan en "la app", piensan en "hablar con el doctor"
- Contexto epidemiológico (pandemia) aceleró adopción tecnológica

Indicadores de Éxito en Emprendimientos Tecnológicos Interculturales

Según Muñoz (2020), medir el éxito requiere métricas claras. En emprendimientos interculturales tech, se requiere visión integral:

Indicadores técnicos:

- Performance de la aplicación (tiempos de carga, uptime)

- Tasa de errores y bugs reportados
- Compatibilidad con dispositivos diversos
- Funcionalidad offline efectiva
- Tiempo de sincronización cuando hay conexión

Indicadores de producto:

- MAU/DAU (Monthly/Daily Active Users)
- Tasa de retención (D1, D7, D30)
- Session duration
- Feature adoption rate
- NPS (Net Promoter Score)

Indicadores de negocio:

- MRR (Monthly Recurring Revenue)
- CAC (Customer Acquisition Cost)
- LTV (Lifetime Value)
- Churn rate
- Tasa de conversión (free to paid)

Indicadores socioculturales:

- Diversidad de usuarios (geografía, lengua, edad)
- Adopción en comunidades target
- Feedback cualitativo de comunidades
- Satisfacción de usuarios diversos
- Reducción de brechas digitales medible

Indicadores de inclusión:

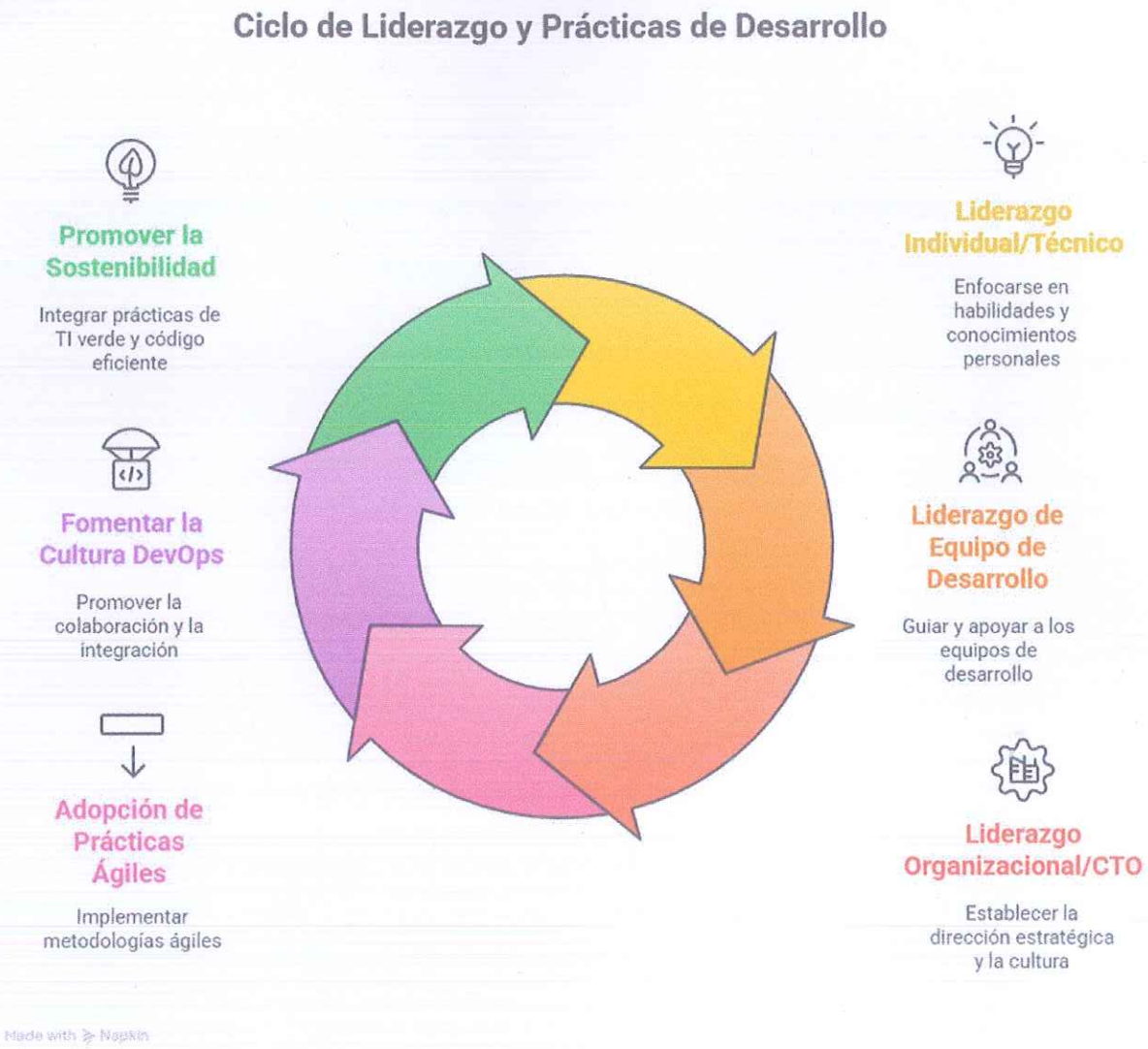
- Porcentaje de equipo de backgrounds diversos
- Representación en user testing y research
- Accesibilidad score (WCAG compliance)
- Disponibilidad en múltiples lenguas
- Uso efectivo por personas con discapacidades

Indicadores de impacto social:

- Usuarios beneficiados en comunidades marginadas
- Mejora medible en KPIs sociales (educación, salud, ingresos)
- Preservación cultural (ej: lenguas, conocimientos)
- Empoderamiento comunitario (capacitaciones, empleos)
- Sostenibilidad ambiental (si aplica)

UNIDAD 2: LIDERAZGO EMPRESARIAL EN DESARROLLO DE SOFTWARE

DIAGRAMA DE APRENDIZAJE



SÍNTESIS

Esta unidad explora el liderazgo como competencia fundamental para el emprendimiento tecnológico exitoso. Se abordan tres niveles de liderazgo: individual (autoconocimiento, desarrollo técnico continuo, liderazgo personal), de equipo (gestión y motivación de equipos de desarrollo, metodologías ágiles, code reviews efectivos) y organizacional (visión técnica estratégica, cultura de ingeniería, escalamiento de equipos tech). Además, se introduce el concepto de desarrollo de software sostenible y economía circular aplicada a tecnología,

paradigmas que los líderes en tech deben integrar para construir empresas tecnológicas responsables, eficientes y resilientes.

Resultado de Aprendizaje: El estudiante será capaz de aplicar principios de liderazgo técnico y empresarial en la gestión de proyectos de desarrollo de software, desarrollando competencias de liderazgo personal y organizacional orientadas a la construcción de equipos de alto rendimiento, culturas de innovación tecnológica, y productos de software sostenibles.

2.1 INDIVIDUOS COMO LÍDERES EN TECNOLOGÍA

Concepto de Liderazgo Técnico

Lussier y Achua (2016) definen el liderazgo como "el proceso de influir en otros para lograr objetivos" (p. 6). En el contexto del desarrollo de software, el liderazgo técnico implica una dimensión adicional: la capacidad de influir mediante expertise técnico, visión de arquitectura, y mentorship en habilidades de programación, sin necesariamente tener autoridad formal.

El liderazgo en tecnología se manifiesta en varios roles:

Tech Lead (Líder Técnico):

- Desarrollador senior que guía decisiones técnicas del equipo
- Define arquitectura, stack tecnológico, patrones de diseño
- Mantiene calidad de código mediante code reviews
- Mentoriza a desarrolladores junior y mid-level
- Balance entre programar (hands-on) y planificar (strategy)

Engineering Manager:

- Gestiona personas más que código directamente
- Contratación, evaluaciones de desempeño, desarrollo de carrera
- Facilita colaboración entre equipos
- Remueve blockers organizacionales
- Traduce visión de producto a planes de ingeniería

CTO (Chief Technology Officer):

- Visión técnica de toda la organización
- Decisiones estratégicas sobre tecnología e infraestructura
- Construye y escala el equipo de ingeniería
- Relación con stakeholders, inversores (pitch técnico)
- Balance entre innovación y estabilidad/deuda técnica

Diferencias entre líder técnico y jefe tradicional:

Líder Técnico	**Jefe Tradicional**
Inspira mediante expertise y ejemplo	Ordena mediante autoridad
Programa junto al equipo	Delega sin involucrarse técnicamente
"Aquí hay un desafío interesante"	"Haz esto"
Genera aprendizaje colectivo	Retiene conocimiento como poder
Code reviews constructivos	Crítica sin contexto
Se ensucia las manos debuggeando	Solo revisa reportes de bugs
Comparte crédito por éxitos	"Mi equipo hizo lo que mandé"
Asume responsabilidad por fallos del sistema	"El desarrollador X falló"

Teorías de Liderazgo Aplicadas a Desarrollo de Software

Basándose en Lussier y Achua (2016), adaptando teorías clásicas al contexto tech:

Teoría Situacional en Tech

No existe un único estilo de liderazgo efectivo; depende de:

- Madurez técnica del equipo: Juniors requieren más dirección, seniors más autonomía
- Complejidad del problema: Bug crítico en producción vs. exploración de nueva feature
- Fase del proyecto: Prototipo rápido vs. refactoring de sistema legacy
- Presión temporal: Demo en 2 días vs. desarrollo planificado

Ejemplo práctico:

- Junior developer en primera semana: Pair programming constante, tareas bien definidas, code review detallado
- Senior developer en problema complejo: Dar contexto y restricciones, dejar autonomía en implementación, revisar solo arquitectura general

Liderazgo Transformacional en Startups Tech

Particularmente relevante en startups donde se requiere:

- Visión inspiradora: "Vamos a democratizar acceso a educación mediante IA"
- Estimulación intelectual: Fomentar exploración de tecnologías emergentes, hackathons internos
- Consideración individualizada: Entender aspiraciones de carrera de cada dev (¿quiere ser tech lead? ¿especialista en ML?)

- Motivación más allá del salary: Impacto del producto, aprendizaje continuo, ownership

Ejemplo: GitHub en sus inicios: Tom Preston-Werner inspiró con visión de "social coding", cultura de trabajo remoto pionera, open source Fridays, autonomía extrema ("be reasonable").

Liderazgo Servicial (Servant Leadership) en Agile

El líder existe para servir al equipo, no viceversa:

- Remover impedimentos: "¿Qué blockers tienen? ¿Necesitan acceso a una API? ¿Problemas con infraestructura?"
- Facilitar colaboración: Organizar knowledge-sharing sessions, facilitar comunicación entre equipos
- Proveer recursos: Asegurar herramientas adecuadas, licencias, hardware, formación
- Proteger al equipo: De interrupciones constantes, feature creep, micromanagement de stakeholders

Manifestación en Scrum: El Scrum Master es líder servicial por excelencia: no dice QUÉ hacer, facilita CÓMO el equipo trabaja mejor.

Autoconocimiento y Desarrollo Personal para Líderes Tech

Según Lussier y Achua (2016), "el autoconocimiento es el punto de partida del desarrollo del liderazgo" (p. 51).

Inteligencia Emocional en Entornos de Alta Presión Tech

Las cinco dimensiones de Goleman aplicadas a desarrollo de software:

1. Autoconciencia:

- Reconocer cuándo estás frustrated por un bug y no tomar decisiones impulsivas
- Saber cuándo necesitas break (burnout prevention)
- Identificar sesgos propios ("prefiero este framework porque lo conozco, no porque sea mejor para el proyecto")
- Consciencia de fortalezas/debilidades técnicas ("soy fuerte en backend, débil en UI, necesito apoyo aquí")

Práctica: Journaling de reflexión post-sprint: ¿Qué me frustró? ¿Qué me energizó? ¿Por qué?

2. Autorregulación:

- No insultar al código de otros en code reviews ("¿WTF is this?")
- Mantener calma cuando el deploy de producción falla a las 11pm
- Resistir urgencia de reescribir todo desde cero (second-system effect)

- Adaptar estilo de comunicación según interlocutor (CEO vs. dev junior)

Práctica: Técnica del STOP:

- Stop: Pausa antes de responder en situación tensa
- Take a breath: Respira profundo
- Observe: ¿Qué estoy sintiendo? ¿Qué necesita el equipo?
- Proceed: Actúa desde lugar de claridad

3. Motivación:

- Impulso por craftsmanship: código limpio, elegante, que funciona
- Compromiso con la misión del producto más allá del paycheck
- Curiosidad por aprender nuevas tecnologías
- Optimismo ante bugs complejos: "es un puzzle interesante"

Señal de motivación intrínseca: Trabajas en side projects por diversión, contribuyes a open source sin compensación, emoción al resolver un algoritmo complejo.

4. Empatía:

- Entender frustración de junior ante error críptico de compilación
- Comprender presión del PM por lanzar feature
- Captar señales no verbales de confusión en pair programming
- Considerar perspectiva de usuario final (no solo elegancia técnica)

Práctica en code reviews: ❌ "Este código es terrible" ☑ "Entiendo la lógica, y hay una forma de hacerlo más eficiente. ¿Quieres que exploremos juntos? Este patrón X podría simplificarlo."

5. Habilidades sociales:

- Explicar conceptos técnicos complejos a no-técnicos
- Negociar plazos realistas con stakeholders
- Networking en conferencias y comunidades tech
- Resolver conflictos: dos devs con visiones opuestas de arquitectura
- Catalizar cambios: convencer al equipo de adoptar nueva herramienta

Herramientas de Autoconocimiento para Desarrolladores

FODA Personal Técnico:

Fortalezas técnicas:

- Lenguajes/frameworks en que soy experto
- Áreas de especialización (backend, frontend, DevOps, ML)
- Soft skills (comunicación, mentorship)

- Experiencias/proyectos exitosos previos

Oportunidades:

- Tecnologías emergentes alineadas con mis intereses
- Roles de liderazgo en mi organización
- Comunidades tech para hacer networking
- Open source projects donde contribuir

Debilidades:

- Áreas técnicas con gaps (ej: "no sé DevOps", "débil en algoritmos")
- Tendencia a sobreestimar plazos (o subestimarlos)
- Dificultad para decir "no" a nuevas features
- Falta de experiencia en liderazgo formal

Amenazas:

- Obsolescencia de habilidades actuales
- Burnout por ritmo insostenible
- Impostor syndrome
- Mercado saturado en mi especialización

Acción: Basado en FODA, crear plan de desarrollo trimestral: "Este trimestre aprenderé Kubernetes mediante proyecto side, buscaré mentor en liderazgo técnico"

Establecimiento de Metas Técnicas y Profesionales

Metodología SMART aplicada a desarrollo de carrera en tech:

Ejemplo de meta NO SMART: "Quiero ser mejor programador"

Ejemplo de meta SMART: "En 6 meses, contribuir a 5 pull requests aceptados en proyecto open source de React, aprender testing avanzado completando curso de Kent C. Dodds, y mentorizar a 1 developer junior en mi equipo mediante sesiones semanales de pair programming"

Áreas clave para metas:

Habilidades técnicas:

- Aprender nuevo lenguaje/framework relevante para objetivos de carrera
- Profundizar en arquitectura de sistemas (microservicios, event-driven)
- Dominar DevOps/CI/CD
- Especializarse en área (ML, blockchain, seguridad)

Habilidades de liderazgo:

- Liderar proyecto técnico de inicio a fin
- Mentor efectivo de X developers

- Presentar en conferencia tech
- Escribir blog técnico con Y posts

Impacto en negocio:

- Entregar feature que incremente métrica clave del producto
- Reducir technical debt en X%
- Mejorar performance del sistema en Y%

Balance vida-trabajo:

- No trabajar fines de semana
- Dedicar Z horas semanales a aprendizaje
- Ejercicio regular para prevenir sedentarismo

Gestión del Tiempo y Productividad para Desarrolladores

El tiempo de un desarrollador es particular: programación requiere "deep work" (trabajo profundo sin interrupciones).

El Costo del Context Switching

Investigaciones muestran que tomar hasta 23 minutos retomar concentración tras interrupción. Para programadores:

- Modelo mental complejo en la cabeza (arquitectura, lógica, edge cases)
- Cada interrupción destruye ese modelo mental
- Necesita tiempo reconstruirlo

Estrategias de protección de deep work:

Time Blocking Técnico:

- Bloques de 2-4 horas sin interrupciones para programación compleja
- "Office hours" específicas para consultas (ej: 2-3pm)
- Mensajes asíncronos (Slack) vs. síncronos (llamadas) cuando sea posible
- Status de "Do Not Disturb" respetado por equipo

Maker Schedule vs. Manager Schedule: Paul Graham diferencia:

- Maker Schedule (desarrolladores): Necesitan bloques largos, una reunión puede "partir" el día
- Manager Schedule: Día dividido en slots de 1 hora, muchas reuniones
- Problema: Managers programan reuniones no entendiendo costo para makers
- Solución: Agrupar reuniones (ej: solo martes y jueves por la tarde), dejar días completos sin reuniones

Técnicas de productividad adaptadas:

Pomodoro para Desarrollo:

- 25 min de coding enfocado
- 5 min break (sin mirar código: estirar, caminar)
- Después de 4 pomodoros, break largo (30 min)
- Beneficio: Previene fatiga mental, burnout

Test-Driven Development (TDD) como time management:

- Escribir test que falla
- Escribir código mínimo para pasar test
- Refactorizar
- Ciclo corto (15-30 min) que da sensación de progreso constante
- Previene "rabbit holes" interminables

Batching de tareas similares:

- Agrupar todos los code reviews en bloque de tiempo
- Responder todos los emails de golpe
- Debugging de múltiples bugs juntos (contexto similar)

Gestión de deuda técnica: Aplicar matriz de Eisenhower:

	URGENTE	NO URGENTE
Importante	Bug crítico en producción (hacer YA)	Refactoring de módulo clave (agendar)
No Importante	Request de feature poco usada (delegar o cuestionar)	Reescribir código "feo" pero funcional (evitar)

Estrategia de deuda técnica: Regla del Boy Scout: "Deja el código un poco mejor de como lo encontraste". Pequeñas mejoras continuas previenen acumulación masiva de deuda.

Valores y Ética en Liderazgo Tecnológico

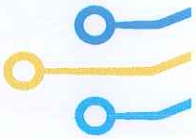
Basándose en Lussier y Achua (2016) sobre ética del liderazgo, aplicado a decisiones tecnológicas:

Valores fundamentales del líder tecnológico:

Integridad del código:

- No copiar código sin licencia apropiada
- Atribución a fuentes (Stack Overflow, tutoriales)
- Rechazar presión por shortcuts que comprometan seguridad
- No inflar capacidades técnicas en pitches/demos

Responsabilidad sobre productos:



- Asumir bugs introducidos ("sí, fué mi merge el que rompió producción")
- No culpar a herramientas, frameworks o infraestructura
- Propiedad (ownership) del código que escribes
- Post-mortems sin culpas, enfocados en sistemic improvements

Respeto en code reviews:

- Comentarios sobre código, nunca sobre personas
- Reconocer que hay múltiples soluciones válidas
- Apreciar esfuerzo aunque resultado no sea óptimo
- Enseñar, no humillar

Privacidad y manejo de datos:

- GDPR, CCPA y leyes locales de protección de datos
- Minimización de datos: recolectar solo lo necesario
- Transparencia con usuarios sobre qué datos se recopilan
- Seguridad por diseño, no como afterthought

Justicia algorítmica:

- Auditar algoritmos por sesgos (género, raza, edad)
- Datasets representativos
- Explicabilidad de decisiones automatizadas
- No optimizar métricas que pueden dañar (engagement adictivo)

Dilemas éticos comunes en desarrollo de software:

Caso 1: Presión por deadline vs. calidad Situación: CEO quiere lanzar feature en 1 semana, pero sabes que hacerlo bien toma 3 semanas. Lanzar rápido significa acumular deuda técnica, bugs potenciales.

Valores en conflicto: Compromiso con stakeholders vs. estándares de calidad

Enfoque ético:

1. Comunicar trade-offs claramente: "Podemos lanzar en 1 semana con estas limitaciones/riesgos, o en 3 semanas con calidad producción-ready"
2. Proponer MVP: ¿Qué subset de feature es viable en 1 semana sin comprometer integridad?
3. Si se decide ir rápido, documentar deuda técnica y agendar tiempo para resolverla
4. Nunca mentir sobre estado real del código

Caso 2: Dark patterns y diseño manipulativo Situación: PM pide implementar feature que dificulta cancelar suscripción (muchos clicks, botones escondidos) para reducir churn.

Valores en conflicto: Métricas de negocio vs. respeto al usuario

Enfoque ético:

1. Cuestionar la táctica: "Esto puede reducir churn corto plazo, pero daña confianza y genera mala reputación"
2. Proponer alternativas: Mejorar producto para que usuarios NO quieran cancelar, no dificultar hacerlo
3. Si hay presión: Escalar a liderazgo, señalar riesgos legales (FTC persigue dark patterns)
4. En última instancia: ¿Estás dispuesto a trabajar en empresa con estas prácticas?

Caso 3: Seguridad vs. conveniencia Situación: Implementar autenticación de dos factores afecta onboarding de usuarios (fricción), pero mejora seguridad dramáticamente.

Valores en conflicto: UX fluida vs. protección de datos de usuarios

Enfoque ético:

1. Educate stakeholders sobre riesgos: "Si hay breach, pérdida de confianza es mucho peor que fricción de signup"
2. Diseño que balancea: 2FA opcional inicialmente, requerida para acciones sensibles, implementación UX cuidadosa
3. Transparencia: Explicar a usuarios POR QUÉ se pide (no solo "por seguridad")
4. Seguir best practices de industria y regulaciones

Toma de decisiones éticas framework:

Ante dilema ético tecnológico, hacerse:

1. ¿Es legal? (regulaciones, licencias, propiedad intelectual)
2. ¿Cómo afecta a usuarios? (privacidad, seguridad, autonomía)
3. ¿Puede escalar negativamente? (pequeño problema ahora, desastre en escala)
4. ¿Pasaría el "newspaper test"? (¿Cómodo si esto sale en portada del periódico?)
5. ¿Alineado con valores declarados de la empresa?
6. ¿Qué diría un mentor que respeto?
7. ¿Consecuencias a largo plazo? (reputación, deuda técnica, precedente)

Código de ética personal para desarrolladores:

Crear un código ético personal que guíe decisiones:

MI CÓDIGO DE ÉTICA COMO DESARROLLADOR

1. CALIDAD: No entregaré código que sé que es defectuoso o inseguro

2. APRENDIZAJE: Admitir cuando no sé algo, y buscar ayuda o aprender

3. HONESTIDAD: Estimaciones realistas, no promesas imposibles

4. RESPETO: Tratar código legacy con respeto; había contexto que desconozco

5. PRIVACIDAD: Tratar datos de usuarios como si fueran de mi familia

6. ACCESIBILIDAD: Cada feature que desarrollo debe ser usable por personas con discapacidades

7. SOSTENIBILIDAD: Escribir código mantenible para mi yo futuro y otros

8. COMUNIDAD: Contribuir a open source, compartir conocimiento

9. BALANCE: No sacrificar salud mental por deadlines

10. IMPACTO: Considerar consecuencias sociales del software que creo

Resiliencia y Gestión del Fracaso en Desarrollo de Software

Según Lussier y Achua (2016), la resiliencia es "la capacidad de recuperarse de la adversidad y crecer a partir de ella" (p. 82). En desarrollo de software, el fracaso es frecuente y educativo.

Tipos de "fracasos" en desarrollo:

Bugs en producción:

- Inevitables a pesar de testing
- Oportunidad de mejorar procesos de QA
- Post-mortem sin culpas, enfocado en prevención sistémica

Features que nadie usa:

- Semanas de desarrollo, 0% adoption
- Aprendizaje sobre validación de hipótesis antes de construir
- Importancia de MVP y testeo temprano

Deuda técnica excesiva:

- Codebase que se vuelve unmaintainable
- Lección sobre balance velocidad vs. calidad
- Necesidad de refactoring proactivo

Startups que fracasan:

- 90% de startups tech fallan
- Aprendizajes sobre product-market fit, timing, ejecución
- Network y experiencia para siguiente emprendimiento

Gestión constructiva del fracaso:

1. Post-Mortem Blameless (sin culpas):

Práctica común en empresas tech maduras como Google, Etsy:

Estructura:

- **¿Qué pasó?** Cronología detallada del incidente
- **¿Por qué pasó?** Root cause analysis (5 whys)
- **¿Qué aprendimos?** Insights para prevenir recurrencia
- **¿Qué haremos diferente**? Action items concretos

Reglas:

- No nombrar individuos como "culpables"
- Enfoque en sistemas, no personas
- Asumir buenas intenciones
- Documentar públicamente para aprendizaje organizacional

Ejemplo real - Bug en producción:

POST-MORTEM: Caída del sistema de pagos - 15 mayo 2024

QUÉ PASÓ:

- 14:32 - Deploy de feature nueva de checkout
- 14:45 - Usuarios reportan errores al pagar
- 14:50 - Equipo identifica bug en validación de tarjetas
- 15:10 - Rollback completado, servicio restaurado
- Impacto: 127 transacciones fallidas, $15,000 en ventas perdidas

POR QUÉ PASÓ:

- Nueva validación de CVV no manejaba formato de ciertos bancos
- ¿Por qué no se detectó en testing?
  - Tests unitarios pasaron (mocks no reflejaban datos reales)
  - ¿Por qué mocks inadecuados?
    - No teníamos ejemplos de todos los formatos de CVV
    - ¿Por qué no?
      - Falta de documentación de edge cases
      - ¿Por qué?
        - No hay proceso de documentar learnings de producción

ROOT CAUSE: Falta de proceso para capturar y documentar edge cases de producción para futuros tests

APRENDIZAJES:

1. Necesitamos staging environment con datos anonimizados reales
2. Tests de integración con pasarela de pagos en sandbox
3. Feature flags para deploys graduales (canary deployment)
4. Monitoring de error rates con alertas automáticas
5. Reencuadre cognitivo del fracaso:

Cambiar narrativa interna:

Pensamiento dañino	**Reencuadre constructivo**
"Soy terrible, rompí producción"	"Cometí un error. ¿Qué puedo aprender? ¿Cómo prevenimos esto sistemáticamente?"
"Esta startup fracasó, soy un fracaso"	"Esta idea no funcionó en este timing con esta ejecución. Tengo aprendizajes valiosos para la siguiente"
"Mi código fue rechazado en code review, no sirvo"	"Recibí feedback para mejorar. Es oportunidad de aprender mejores prácticas"
"No puedo resolver este bug, soy mal developer"	"Este bug es complejo. ¿Qué recursos necesito? ¿Quién puede ayudarme?"

3. Cultura que celebra fallos rápidos:

Empresas tech innovadoras como Amazon, Netflix:

- "Fail fast, learn faster"
- Experimentos baratos para validar antes de comprometer recursos
- Blameless post-mortems públicos
- "Failure is not falling down, but refusing to get up"

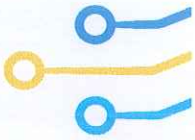
Práctica concreta: "Demo de Fracasos" mensual donde equipos comparten:

- Qué experimentamos
- Qué falló
- Qué aprendimos
- Celebración (literal aplausos) por aprendizaje

4. Resiliencia técnica (del sistema) y personal (del individuo):

Resiliencia del sistema:

- Chaos engineering (Netflix Simian Army)



- Redundancia, fallbacks, graceful degradation
- Diseñar asumiendo que cosas fallarán

Resiliencia personal:

- Rutinas de autocuidado (ejercicio, sueño, desconexión)
- Red de soporte (mentores, peers, comunidad)
- Propósito claro más allá de cualquier proyecto específico
- Recordar éxitos pasados en momentos de duda

Síndrome del impostor en tecnología:

Particularmente prevalente en tech debido a:

- Velocidad de cambio tecnológico (siempre hay algo que no sabes)
- Cultura de "rockstar developers" en medios
- Comparación con perfiles curados en LinkedIn/Twitter
- Contribuciones open source públicas (tu código expuesto)

Señales:

- Atribuir éxitos a suerte, fracasos a falta de capacidad
- Miedo constante de ser "descubierto" como fraude
- Perfectionism paralizante
- Evitar desafíos por miedo a fallar

Estrategias de manejo:

1. Nombrar el fenómeno: Saber que es común (70% de personas lo experimentan)
2. Hablar de ello: Compartir con peers, descubrir que otros también lo sienten
3. Documentar logros: Lista de wins, proyectos completados, problemas resueltos
4. Redefinir expertise: No saberlo todo, sino saber cómo aprender y resolver
5. Mentorizar: Enseñar a otros demuestra tu conocimiento
6. Comunidad: Nadie sabe todo; todos tienen gaps

2.2 LIDERAZGO DE EQUIPO Y ORGANIZACIONAL EN DESARROLLO DE SOFTWARE

Formación y Desarrollo de Equipos de Desarrollo

Según Lussier y Achua (2016), los equipos pasan por etapas predecibles. En equipos de desarrollo de software:

Etapas de desarrollo de equipos técnicos (Modelo de Tuckman adaptado):

1. Formación (Forming) - "Setup"

Características en equipos de desarrollo:

- Nuevo equipo o nuevos miembros se integran
- Incertidumbre sobre: stack tecnológico, arquitectura, procesos (Git flow, code review)
- Comportamiento educado, exploratorio
- Todos hacen preguntas básicas
- Dependencia del líder técnico para dirección

Necesidades del equipo:

- Claridad sobre proyecto, objetivos, tech stack
- Definición de roles (quién frontend, backend, DevOps)
- Establecer normas de código (linter, formatter, style guide)
- Setup de desarrollo (repos, ambientes, accesos)
- Kick-off técnico: arquitectura overview

Rol del líder técnico:

- Documentar decisiones técnicas y comunicarlas claramente
- Pair programming para transmitir contexto
- Crear README comprehensivo y docs de arquitectura
- Configurar CI/CD y herramientas desde día uno
- Facilitar conocimiento entre miembros (icebreakers técnicos: "¿Tu lenguaje/framework favorito?")

Ejemplo de actividad: "Architecture Decision Records" (ADRs): Documentar por qué elegimos React vs. Vue, PostgreSQL vs. MongoDB, con pros/cons considerados. Nuevo miembro puede entender contexto de decisiones pasadas.

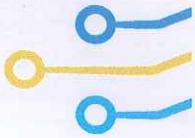
2. Conflicto (Storming) - "Merge Conflicts"

Características en equipos de desarrollo:

- Emergen diferencias en enfoques técnicos (tabs vs. spaces, OOP vs. functional)
- Desacuerdos sobre arquitectura, patrones de diseño
- Frustración si codebase legacy es caótico
- Cuestionamiento de decisiones técnicas del líder
- Competencia por áreas del código ("quiero trabajar en lo interesante, no en CRUD aburrido")

Conflictos técnicos comunes:

- "Deberíamos migrar a microservicios" vs. "Monolito está bien por ahora"
- "Necesitamos más tests" vs. "Tests nos ralentizan"
- "Refactor completo" vs. "Funciona, no lo toques"



- Code style preferences personales

Rol del líder técnico:

- Facilitar debates técnicos estructurados (RFCs, architecture reviews)
- Establecer criterios objetivos para decisiones (performance benchmarks, métricas)
- Crear espacio seguro para desacuerdos técnicos
- Mediar conflictos: Enfocarse en qué es mejor para el producto/usuarios, no egos
- Establecer convenciones de equipo (linter automático resuelve tabs vs. spaces)

Técnicas de resolución:

Architecture Decision Records (ADRs):

Markdown

ADR-005: Elegir base de datos

Contexto

Necesitamos persistir datos de usuarios, transacciones, productos.

Volumen esperado: 10K usuarios en primer año, 1M en 3 años.

Opciones consideradas

1. PostgreSQL (relacional)

2. MongoDB (documental)

3. DynamoDB (NoSQL managed)

Decisión

PostgreSQL

Razones

- Transacciones ACID críticas para pagos
- Estructura de datos relativamente fija
- Team tiene más experiencia con SQL
- Hosting en AWS RDS facilita escalado
- Puede migrar a DynamoDB después si necesario

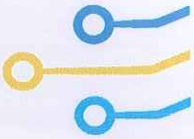
Consecuencias

- Aprender Sequelize ORM
- Backups automáticos vía RDS
- Costo predecible

Fecha: 2024-05-15

Decidido por: Tech Lead (María) + equipo (consenso)

Tech Spikes para validar opciones: Cuando hay desacuerdo técnico, hacer spike (timeboxed experiment):



- 2 días implementando POC de opción A
- 2 días implementando POC de opción B
- Decidir basándose en evidencia, no opiniones

3. Normalización (Norming) - "Established Workflow"

Características:

- Se establecen normas de trabajo y código
- Mayor cohesión, comienza confianza
- Roles más claros (frontend specialist, backend specialist)
- Code reviews constructivos, no defensivos
- Apreciación de fortalezas diversas del equipo

Normas técnicas establecidas:

- Git workflow definido (Gitflow, trunk-based, etc.)
- Proceso de code review claro (quién revisa, criterios, turnaround time)
- Definition of Done acordada
- Horarios de disponibilidad (para trabajo remoto)
- Canales de comunicación (cuándo Slack, cuándo meeting, cuándo asíncrono)

Ejemplo de normas documentadas:

markdown

TEAM WORKING AGREEMENT - Equipo Backend

Code Reviews

- Todo PR requiere mínimo 1 aprobación antes de merge
- Reviews dentro de 24 horas (workdays)
- Comentarios siempre constructivos, sugiriendo alternativas
- Prefijo en comentarios: [BLOCKER] vs [SUGGESTION] vs [QUESTION]
- Aprobar con "LGTM" + comentario específico de qué te gustó

Git Workflow

- Branch naming: feature/TICKET-123-description
- Commits: Conventional commits (feat:, fix:, docs:)
- Squash commits antes de merge
- Delete branch después de merge

Testing

- Unit tests para toda lógica de negocio
- Integration tests para endpoints críticos
- Coverage mínimo 70%

- Tests deben pasar en CI antes de merge

Communication

- Daily async standup en Slack (9-10am)
- Code questions: Slack #backend-help
- Blockers urgentes: Tag @tech-lead
- Deep work hours: 2-5pm (minimize interruptions)
- Pair programming: Agendar, no sorpresivo

On-Call Rotation

- Rotación semanal
- Responder incidentes de producción
- Documentar en post-mortem
- Compensación: Time off equivalente

Rol del líder técnico:

- Reforzar comportamientos positivos ("Excelente code review, Juan, constructivo y educativo")
- Crear rituales de equipo (demo Fridays, tech talks internos)
- Empoderar al equipo (delegar decisiones técnicas pequeñas)
- Celebrar wins (feature shipped, bug complejo resuelto)

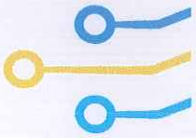
4. Desempeño (Performing) - "Flow State

Características:

- Alta productividad
- Autonomía del equipo (poco micromanagement necesario)
- Resolución eficiente de problemas técnicos
- Pair programming fluido
- Code reviews rápidos y efectivos
- Enfoque en objectives, no en horas

Señales de equipo en performing:

- Velocity de sprint consistente y predecible
- Proactividad: developers identifican problemas antes de que se asignen
- Knowledge sharing orgánico (slack threads, docs)



- Debugging colaborativo ("¿Alguien vio este error?")
- Pride en el código del equipo

Rol del líder técnico:

- Mantener momentum (celebrar logros, mantener visión)
- Eliminar blockers rápidamente
- Proteger al equipo de distracciones externas
- Proveer desafíos técnicos (no complacencia)
- Mantener visión a largo plazo (no solo feature churn)

Prácticas de equipos de alto desempeño:

Mob Programming para problemas complejos:

- Todo el equipo trabaja en el mismo problema simultáneamente
- Una persona al teclado (driver), otros guían (navigators)
- Rotan cada 10-15 min
- Para: arquitectura compleja, bugs críticos, onboarding

Tech Debt Fridays:

- Un día a la semana dedicado a mejorar codebase
- No features nuevas
- Refactoring, optimización, tests, documentación
- Previene acumulación de deuda

Blameless post-mortems públicos:

- Transparencia sobre failures
- Aprendizaje organizacional
- Cultura que no teme admitir errores

5. Cierre/Duelo (Adjourning) - "Deployment & Handoff"

Características en tech:

- Proyecto se completa o equipo se disuelve
- Transición de código a otro equipo (o a mantenimiento)
- Reflexión sobre logros y aprendizajes

- Emociones mixtas (orgullo, melancolía si fue buen equipo)

Actividades de cierre:

- Retrospectiva final comprehensiva
- Documentación completa para handoff
- Celebration (demo pública, testimonios)
- Reconocimiento individual de contribuciones
- Mantener contacto (LinkedIn, alumni Slack)

Construcción de Equipos de Alto Rendimiento en Tech

Basándose en Lussier y Achua (2016), características de equipos efectivos aplicadas a desarrollo de software:

1. Objetivos claros y compartidos:

En desarrollo de software:

- **Product goals:** Qué construimos, para quién, por qué
- **Technical goals:** Performance targets, uptime, code quality
- **Team goals:** Velocity, cycle time, code review turnaround

Ejemplo OKR (Objectives and Key Results) de equipo:

OBJETIVO: Mejorar experiencia de onboarding de usuarios

3. Comunicación abierta y efectiva:

Desafíos de comunicación en equipos de desarrollo:

- Trabajo asíncrono (remote, zonas horarias)
- Comunicación escrita predominante (PRs, Slack, docs)
- Necesidad de deep work sin interrupciones
- Equipos distribuidos globalmente

Best practices de comunicación técnica:

Documentación como comunicación asíncrona:

- READMEs detallados en repos

- Architecture Decision Records (ADRs)
- API documentation (Swagger, Postman collections)
- Runbooks para operaciones comunes
- Onboarding guides

4. Confianza mutua:

En equipos de desarrollo, confianza significa:

Confiabilidad técnica:

- "Si María dice que lo hará, lo hará"
- Cumplir commits (o comunicar temprano si hay delay)
- Code que funciona (no romper main/producción)
- Responder a on-call cuando es tu turno

Competencia técnica:

- "Confío en que Juan puede resolver esto"
- Code reviews honestos pero constructivos
- Pedir ayuda sin vergüenza
- Compartir conocimiento sin gatekeeping

Integridad:

- "Sara es honesta sobre qué sabe y qué no"
- Estimaciones realistas (no sobre-prometer)
- Admitir errores rápidamente
- Transparencia sobre dificultades

Benevolencia:

- "Pedro realmente quiere que yo crezca"
- Mentorship generoso
- Compartir crédito
- Pair programming sin juzgar

Cómo construir confianza en equipos tech:

- Pair programming regular: Trabajar juntos, ver proceso de pensamiento
- Vulnerability: Líder que admite "no sé" modela safety psicológica

- Consistency: Cumplir lo que dices, repetidamente
- Small wins together: Celebrar
logros colectivos

5. Gestión efectiva de conflictos:

Conflictos técnicos comunes y resolución:

Desacuerdo sobre arquitectura:

- Facilitación: Ambos presentan propuesta (diagramas, pros/cons)
- Data: Hacer POCs pequeños, medir (performance, complejidad)
- Timeboxing: Si no hay consenso en X tiempo, tech lead decide
- Document: ADR con razones de decisión

Frustración con codebase legacy:

- Empatía: Validar frustración ("sí, este código es difícil de trabajar")
- Plan: Refactoring incremental, no rewrite completo
- Boy Scout rule: Mejorarlo poco a poco
- Perspective: Código legacy = código que funciona y genera value

Carga de trabajo desbalanceada:

- Visibility: Hacer trabajo visible (Kanban board)
- Retrospectivas: Espacio seguro para airear frustraciones
- Rotación: Rotar tareas menos deseables (on-call, legacy maintenance)
- Recognition: Agradecer públicamente trabajo menos glamoroso

6. Liderazgo compartido:

En equipos técnicos maduros, liderazgo se distribuye:

- Arquitectura**: Senior más experto en área específica
- Procesos: Developer que mejora workflows (CI/CD, automation)
- Calidad: Champion de testing y code quality
- Mentorship: Varios seniors mentorizan juniors
- Producto: Todos contribuyen a product thinking

Ejemplo: Guild model (Spotify):

- Squads: Equipos autónomos con misión específica

- Chapters: Developers de misma especialidad across squads (Backend chapter)
- Guilds: Comunidades de interés (Testing guild, Security guild)
- Liderazgo emerge según expertise y contexto

7. Mejora continua:

Prácticas de mejora continua en equipos de desarrollo:

Retrospectivas efectivas:

Formato "Start, Stop, Continue":

RETROSPECTIVA - Sprint 23

START (qué comenzar a hacer):

- Pair programming para features complejas
- Documentar decisiones técnicas en ADRs
- Slack thread semanal de TIL (Today I Learned)

STOP (qué dejar de hacer):

- Meetings sin agenda
- Aprobar PRs sin realmente reviewear
- Working overtime regularmente (burnout alert)

CONTINUE (qué seguir haciendo):

- Tech talks internos viernes (muy útiles)
- Code reviews constructivos
- Celebrating wins en #general

ACTION ITEMS:

- Juan organizará calendario de pair programming
- María creará template de ADR en docs
- Pedro iniciará TIL thread este viernes

Métricas de equipo (DORA metrics):

- Deployment frequency: Qué tan seguido deployamos
- Lead time for changes: Tiempo de commit a producción
- Time to restore service: Qué tan rápido arreglamos incidents
- Change failure rate: % de deploys que requieren rollback/hotfix

Meta: Mejorar estas métricas incrementalmente

Learning culture:

- Tech talks internos: Developer presenta algo que aprendió (30min, viernes)

- Book club técnico: Leer y discutir libro juntos ("Clean Code", "Designing Data-Intensive Applications")
- Hack days: Trimestral, 2 días para experimentar con tech nueva
- Conference budgets: Enviar developers a conferencias relevantes

Diversidad en Equipos de Desarrollo

Lussier y Achua (2016) destacan que "equipos diversos superan a equipos homogéneos en creatividad e innovación cuando hay inclusión efectiva" (p. 387).

Tipos de diversidad relevantes en tech:

Diversidad técnica:

- Backgrounds diversos (CS degree, bootcamp, autodidacta)
- Especializaciones (frontend, backend, mobile, data)
- Experiencia (junior, senior, diferentes industrias previas)
- Stack preferences (Java vs. Python, SQL vs. NoSQL)

Beneficio: Soluciones más robustas, menos puntos ciegos técnicos

Diversidad cognitiva:

- Pensadores visuales vs. abstractos
- Top-down vs. bottom-up
- Big picture vs. detalles
- Risk-takers vs. cautelosos

Beneficio: Balance entre innovación y estabilidad

Diversidad demográfica:

- Género, edad, origen étnico, nacionalidad
- Neurodiversidad (ADHD, autismo, etc.)
- Orientación sexual, identidad de género

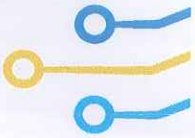
Beneficio: Productos más inclusivos, menor sesgo en diseño

Gestión efectiva de diversidad en equipos tech:

Inclusión activa:

- Todos invited a contribuir en decisiones técnicas
- Rotar quién presenta en demos
- Valorar explícitamente perspectivas diversas ("Buena pregunta, no había considerado ese ángulo")
- Interrupciones: Moderador asegura que todos puedan hablar

Mitigar sesgos inconscientes:



- Blind resume screening en hiring
- Structured interviews (mismas preguntas a todos)
- Diverse interview panels
- Calibración regular de evaluaciones de performance

Crear psychological safety:

- Normalizar "no sé" y preguntas
- Líder modela vulnerabilidad
- Cero tolerancia a condescendencia, mansplaining
- Celebrar contribuciones de todos, no solo "rockstars"

Acomodar necesidades diversas:

- Flexibilidad horaria (padres, cuidadores, neurodiversidad)
- Work-from-home options
- Comunicación asíncrona (para introverts, non-native speakers)
- Herramientas de accesibilidad

2.3 ECONOMÍA CIRCULAR Y SOSTENIBILIDAD EN DESARROLLO DE SOFTWARE

Concepto de Economía Circular Aplicada a Software

La economía circular busca mantener recursos en uso el máximo tiempo posible. En software:

Modelo lineal tradicional en tech:

1. Desarrollar código
2. Usar código
3. Código se vuelve legacy u obsoleto
4. Reescribir desde cero

Problemas:

- Desperdicio de esfuerzo de desarrollo
- Knowledge loss cuando reescribes
- Deuda técnica acumulada

Modelo circular en software:

- **Reducir:** Minimizar recursos computacionales usados
- **Reusar:** Código modular, DRY (Don't Repeat Yourself)
- **Refactorizar:** Mejorar código existente en lugar de reescribir
- **Reciclar:** Open source, compartir componentes

- **Regenerar:** Contribuir al ecosistema (documentación, bug fixes)

Green Software Engineering (Software Sostenible)

Movimiento emergente que reconoce impacto ambiental del software.

Impacto ambiental del software:

Consumo energético de datacenters:

- Datacenters consumen 1% de electricidad global
- Emisiones de CO2 de IT ~ industria aérea
- Cooling de servidores consume energía masiva

Dispositivos de usuarios:

- Software ineficiente drena baterías más rápido
- Obsolescencia programada (software que requiere hardware nuevo)
- E-waste de dispositivos descartados

UNIDAD 3: EMPRENDIMIENTO E INNOVACIÓN EN DESARROLLO DE SOFTWARE

DIAGRAMA DE APRENDIZAJE

[Imagen sugerida: Representación visual del proceso de Design Thinking (empatizar, definir, idear, prototipar, testear) integrado con el ciclo de desarrollo de software (requisitos, diseño, implementación, testing, deployment), mostrando diferentes tipos de emprendimientos tech y fuentes de innovación como IA, blockchain, cloud computing]

SÍNTESIS

Esta unidad profundiza en los conceptos y metodologías de emprendimiento e innovación específicamente aplicados al desarrollo de software. Se exploran metodologías ágiles de innovación como Design Thinking y Lean Startup, fundamentales para crear productos digitales centrados en el usuario. Se analizan los factores que facilitan o inhiben el emprendimiento tecnológico en el contexto ecuatoriano, se estudian diferentes tipos de emprendimientos en la industria del software, y se identifican las fuentes de innovación tecnológica que pueden impulsar nuevos proyectos. El objetivo es equipar a los estudiantes con herramientas prácticas para generar, validar e implementar ideas innovadoras de productos de software que resuelvan problemas reales y generen valor en el mercado.

Resultado de Aprendizaje: El estudiante será capaz de desarrollar propuestas innovadoras de software utilizando metodologías de emprendimiento e innovación tecnológica, identificando oportunidades en diversos sectores, aplicando Design Thinking y Lean Startup para validación temprana, y aprovechando fuentes diversas de innovación para crear soluciones viables, escalables y centradas en el usuario.

3.1 CONCEPTUALIZACIÓN DE EMPRENDIMIENTO E INNOVACIÓN - DESIGN THINKING

Relación entre Emprendimiento e Innovación en Software

Según Osterwalder y Pigneur (2010), "la innovación en modelos de negocio consiste en crear valor de nuevas formas" (p. 14). En el contexto del desarrollo de software:

Emprendimiento en Software:

- Proceso de identificar oportunidades tecnológicas
- Crear y lanzar productos digitales
- Construir organizaciones que escalen
- Puede ser innovador o replicar modelos existentes

Innovación en Software:

- Creación e implementación de soluciones tecnológicas novedosas
- Puede ocurrir en startups o empresas establecidas
- Nuevas formas de resolver problemas mediante código
- No necesariamente implica crear empresa nueva

Emprendimiento innovador en Tech:

- Combinación poderosa: nuevas empresas basadas en innovación tecnológica
- Mayor potencial de crecimiento exponencial
- Ejemplos: Google (innovación en search), Uber (innovación en matching de oferta/demanda), Stripe (innovación en pagos)

Según Muñoz (2020), "las startups de mayor impacto combinan soluciones tecnológicas innovadoras con modelos de negocio escalables" (p. 67).

Tipos de Innovación en Desarrollo de Software

Basándose en Prieto (2017) sobre clasificación de innovaciones:

Según objeto de innovación:

Innovación de producto (Software como producto):

- Nuevas aplicaciones, plataformas, herramientas
- Mejoras significativas en software existente
- Nuevas features que transforman experiencia

Ejemplos:

- **Radical:** Notion (reimaginar documentos/bases de datos)
- **Incremental:** GitHub Copilot (mejora sobre IDEs existentes)
- **Disruptiva:** Figma (design colaborativo en browser vs desktop apps)

Innovación de proceso (Cómo desarrollamos):

- Nuevas metodologías de desarrollo (DevOps, GitOps)
- Herramientas que mejoran productividad de developers
- Automatización de testing, deployment, monitoring

Ejemplos:

- CI/CD pipelines (GitHub Actions, GitLab CI)

- Infrastructure as Code (Terraform, Pulumi)
- No-code/Low-code platforms (Bubble, Webflow)

Innovación de modelo de negocio:

- Nuevas formas de monetizar software
- Cambios en cómo se entrega valor a clientes

Ejemplos:

- SaaS (Salesforce pionero en "software as service")
- Freemium (Dropbox, Spotify)
- API-first business (Stripe, Twilio)
- Open source con servicios (Red Hat, MongoDB)

Innovación de arquitectura:

- Nuevas formas de estructurar sistemas
- Paradigmas de desarrollo emergentes

Ejemplos:

- Microservicios vs. monolito
- Serverless computing (AWS Lambda)
- Edge computing
- Jamstack architecture

Según grado de novedad:

Innovación incremental:

- Mejoras graduales sobre soluciones existentes
- Menor riesgo, más frecuente
- 80% de innovación en software

Ejemplo: Agregar dark mode a tu app, implementar autenticación biométrica, mejorar algoritmo de recomendación 10%

Innovación radical:

- Cambios fundamentales que crean nuevas categorías
- Mayor riesgo, mayor potencial de impacto
- Requiere educación de mercado

Ejemplo: Blockchain (nueva forma de verificar transacciones), React (nuevo paradigma de UI components), Docker (nueva forma de empaquetar aplicaciones)

Innovación disruptiva (Clayton Christensen):

- Inicialmente sirve mercado pequeño o no atendido
- Calidad/features menores que incumbentes
- Pero con otras ventajas (precio, accesibilidad, conveniencia)
- Eventualmente mejora y desplaza a incumbentes

Ejemplo clásico:

- AWS disrumpió datacenters tradicionales:
    - Inicialmente: Menos confiable, menos features
    - Pero: Mucho más barato, self-service, pago por uso
    - Eventualmente: Mejoró hasta superar en features
    - Resultado: Dominó el mercado, transformó industria

Según fuente:

Innovación cerrada (tradicional):

- I+D interno exclusivamente
- Control total, pero recursos limitados
- Común en corporaciones grandes (Microsoft, Apple tradicionalmente)

Innovación abierta:

- Colaboración con externos
- Open source, APIs públicas, developer communities
- Acceso a más ideas, talento, velocidad

Ejemplo: Google con Android (código abierto), Meta con React, Microsoft con VSCode

Design Thinking para Desarrollo de Software

Según Muñoz (2020), Design Thinking es "una metodología centrada en el usuario para resolver problemas complejos mediante empatía, creatividad e iteración" (p. 89).

¿Por qué Design Thinking en Software?

Problemas comunes en desarrollo de software sin Design Thinking:

- Construir features que nadie usa
- Interfaces confusas para usuarios
- Soluciones técnicamente elegantes pero que no resuelven problema real
- Falta de validación antes de invertir meses de desarrollo

Design Thinking mitiga esto mediante enfoque en usuario desde el inicio.

Principios fundamentales de Design Thinking:

1. Human-centered (Centrado en el humano):

- Empatía profunda con usuarios finales
- Entender necesidades, no solo requerimientos declarados
- Usuarios como co-creadores, no solo testers finales

2. Show, don't tell:

- Prototipar temprano y frecuentemente
- Hacer tangible para obtener feedback real
- Aprender haciendo, no solo planificando

3. Radical collaboration:

- Equipos multidisciplinarios (devs, designers, PMs, usuarios)
- Diversidad de perspectivas genera mejores soluciones

- Co-creación con stakeholders

4. Optimismo consciente:

- Creer que hay soluciones
- Experimentación sin miedo al fracaso
- Cada problema es oportunidad de innovar

5. Iteración:

- Proceso no lineal
- Fallar rápido y barato para aprender
- Mejorar continuamente basándose en feedback

Las Cinco Etapas del Design Thinking Aplicadas a Software

ETAPA 1: EMPATIZAR

Objetivo: Comprender profundamente a usuarios, sus contextos, problemas y necesidades.

Métodos de investigación para productos de software:

User Interviews (Entrevistas de usuario):

Preparación:

- Reclutar usuarios representativos de tu target
- Preparar guía de entrevista (temas, no script rígido)
- Ubicación: Donde usuario está cómodo
- Grabar (con permiso) para análisis posterior

Técnicas de entrevista:

- **Preguntas abiertas:** "Cuéntame sobre la última vez que intentaste [hacer X]"
- **5 Whys:** Profundizar en motivaciones. "¿Por qué eso es importante?" → "¿Por qué?" (repetir)
- **Observar emociones:** Frustración, entusiasmo en tono de voz
- **Escuchar activamente:** No interrumpir, validar, profundizar

Ejemplo de entrevista para app de productividad:

ENTREVISTADOR: Cuéntame sobre cómo organizas tu trabajo actualmente.

USUARIO: Uso una combinación de Google Calendar, Trello, y muchos papelitos...

E: ¿Por qué usas múltiples herramientas?

U: Calendar es para meetings, Trello para proyectos grandes... pero los papelitos son para recordatorios inmediatos porque es más rápido.

E: ¿Qué sería "más rápido" en una app digital?

U: Algo que pueda abrir y escribir en segundos, sin tener que navegar menús o categorizar. Como literalmente pensamiento → nota.

E: ¿Qué pasa con esos papelitos después?

U: [risas] Se pierden, se ensucian con café... es caótico. Pero es que las apps son demasiado complicadas para cosas simples.

E: Cuéntame de una vez que perdiste algo importante en un papelito...

[INSIGHT: Fricción en captura rápida es crítica. Usuario valora speed sobre organización para capture, pero necesita organización después]

Contextual Inquiry (Observación en contexto):

Observar a usuarios en su ambiente real utilizando soluciones actuales.

Proceso:

- Shadowing: Seguir al usuario en su día típico
- Ask them to show you: "Muéstrame cómo haces X"
- Observe painpoints: Workarounds, frustraciones, pasos extra

Ejemplo: Observar a vendedor usando CRM actual:

- Abre 4 tabs diferentes para ver info completa de cliente
- Copia-pega datos entre sistemas
- Pierde 5 minutos buscando email de hace 2 meses
- → **Insight:** Necesita vista unificada del cliente, search potente

User Personas (Basadas en datos reales):

Representaciones de usuarios arquetípicos basadas en investigación real.

Componentes de Persona para software:

markdown

PERSONA: "Carlos el Developer Freelancer"

Demografía

- Edad: 28
- Ubicación: Quito, Ecuador
- Ocupación: Full-stack developer freelance
- Ingresos: $2,000/mes (variable)
- Educación: Autodidacta + bootcamp

Contexto Técnico

- Stack: React, Node.js, PostgreSQL, AWS
- Herramientas: VSCode, GitHub, Figma, Notion
- Experiencia: 4 años
- Setup: Laptop personal, trabaja desde casa/cafés

Comportamiento

- Trabaja 6-8 horas/día (horario flexible)
- 3-5 proyectos simultáneos
- Busca clientes en Upwork, LinkedIn
- Usa YouTube/blogs para aprender tecnologías nuevas

Objetivos

- Aumentar ingresos a $3,500/mes
- Ganar clientes recurrentes (no solo one-off projects)
- Automatizar partes administrativas (invoicing, tracking time)
- Mejorar portfolio para conseguir mejores clientes

Frustraciones

- Mucho tiempo en admin (emails, invoices, contratos)
- Difícil estimar proyectos accuradamente
- Miedo de no conseguir siguiente cliente
- Soledad del freelancing (extraña team)
- Imposter syndrome cuando aprende tech nueva

Necesidades de Software

- Gestión de proyectos simple (no Jira enterprise)
- Time tracking automático
- Invoicing con plantillas ecuatorianas
- Portfolio builder
- Comunidad de otros freelancers

UNIDAD 4: MODELO DE NEGOCIO Y MARKETING DIGITAL

4.1 GENERACIÓN DE IDEAS DE NEGOCIOS

Desarrollo: La generación de ideas de negocios inicia con la observación activa del entorno para detectar problemas no resueltos o necesidades insatisfechas. Se trabaja con técnicas como el análisis de tendencias, entrevistas breves, inspiración en modelos existentes y metodologías creativas como SCAMPER, que permiten modificar productos o servicios ya conocidos. Se motiva al estudiante a pensar en soluciones innovadoras y sostenibles.

Resultados esperados: capacidad para identificar oportunidades, crear propuestas iniciales y justificar su relevancia.

4.2 ESTRUCTURA DEL MODELO DE NEGOCIO

Desarrollo: Se profundiza en el uso del **modelo Canvas** como herramienta visual para estructurar una idea de negocio. Cada bloque es analizado mediante ejemplos reales y ejercicios guiados. Se enfatiza la relación entre la propuesta de valor y el segmento de clientes, explicando cómo ambos deben encajar de manera armoniosa para generar impacto.

Resultados esperados: manejo del Canvas completo, claridad en la propuesta de valor y coherencia entre costos e ingresos.

4.3 DESARROLLO DEL MODELO DE NEGOCIO

Desarrollo: En esta etapa se pasa del diseño a la validación. Se enseña a realizar entrevistas con clientes potenciales, identificar supuestos críticos y construir un prototipo mínimo viable (MVP). Los estudiantes comparan sus hipótesis iniciales con los resultados obtenidos y ajustan su modelo. Se fomenta una visión iterativa y flexible.

Resultados esperados: modelo ajustado, prototipo básico y registro de aprendizajes.

4.4 FUNDAMENTOS DEL PLAN DE MARKETING DIGITAL

Desarrollo: Se introduce el marketing digital como un conjunto de estrategias para posicionar un negocio en medios online. Se profundiza en el embudo digital (atracción, interacción, conversión y fidelización) y se explica el funcionamiento de herramientas como SEO, anuncios pagados, marketing de contenidos y email marketing. Los estudiantes diseñan un plan sencillo adaptado a su idea de negocio.

Resultados esperados: plan de marketing digital inicial con objetivos claros y estrategias básicas.

4.5 PLATAFORMA DE VENTA DIGITAL

Desarrollo: Se estudian los principales medios digitales para comercializar productos: redes sociales (Facebook, Instagram, TikTok), marketplaces y plataformas de e-commerce. Se enseña a crear perfiles comerciales, optimizar publicaciones, usar hashtags, integrar métodos de pago y gestionar la experiencia del cliente.

Resultados esperados: creación de una página de venta funcional y publicación optimizada.

4.6 HERRAMIENTAS DE MEDICIÓN, AUTOMATIZACIÓN Y MONITORIZACIÓN

Desarrollo: El estudiante aprende a interpretar métricas clave como alcance, clics, conversiones y tasa de interacción. Se trabaja con herramientas como Meta Business Suite, Google Analytics y Search Console. Además, se introducen sistemas de automatización como chatbots, secuencias de correo y programadores de contenido.

Resultados esperados: reporte básico de métricas, análisis de datos y mejoras sugeridas.

ELABORACIÓN, REVISIÓN Y APROBACIÓN DE PARES	
Profesor	
Ing. Betty Alexandra Jaramillo Tituaña; MEd **Fecha de elaboración:** 23/12/2025	
Comisión de revisión de pares de guías de estudio del Instituto Superior Tecnológico Tena	
Lcda. María Angélica Campoverde Encalada	Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango	Mg. Duarte Mora Martha Janina
Abg. Danilo Alexander Zamora Núñez., Mg. **Fecha de revisión:** 26/12/2025	
Coordinador de Investigación, Desarrollo Tecnológico e Innovación	
Abg. Danilo Alexander Zamora Núñez., Mg. **Fecha de aprobación:** 29/12/2025	

INSTITUTO SUPERIOR TECNOLÓGICO TENA
COORDINACIÓN DE INVESTIGACIÓN