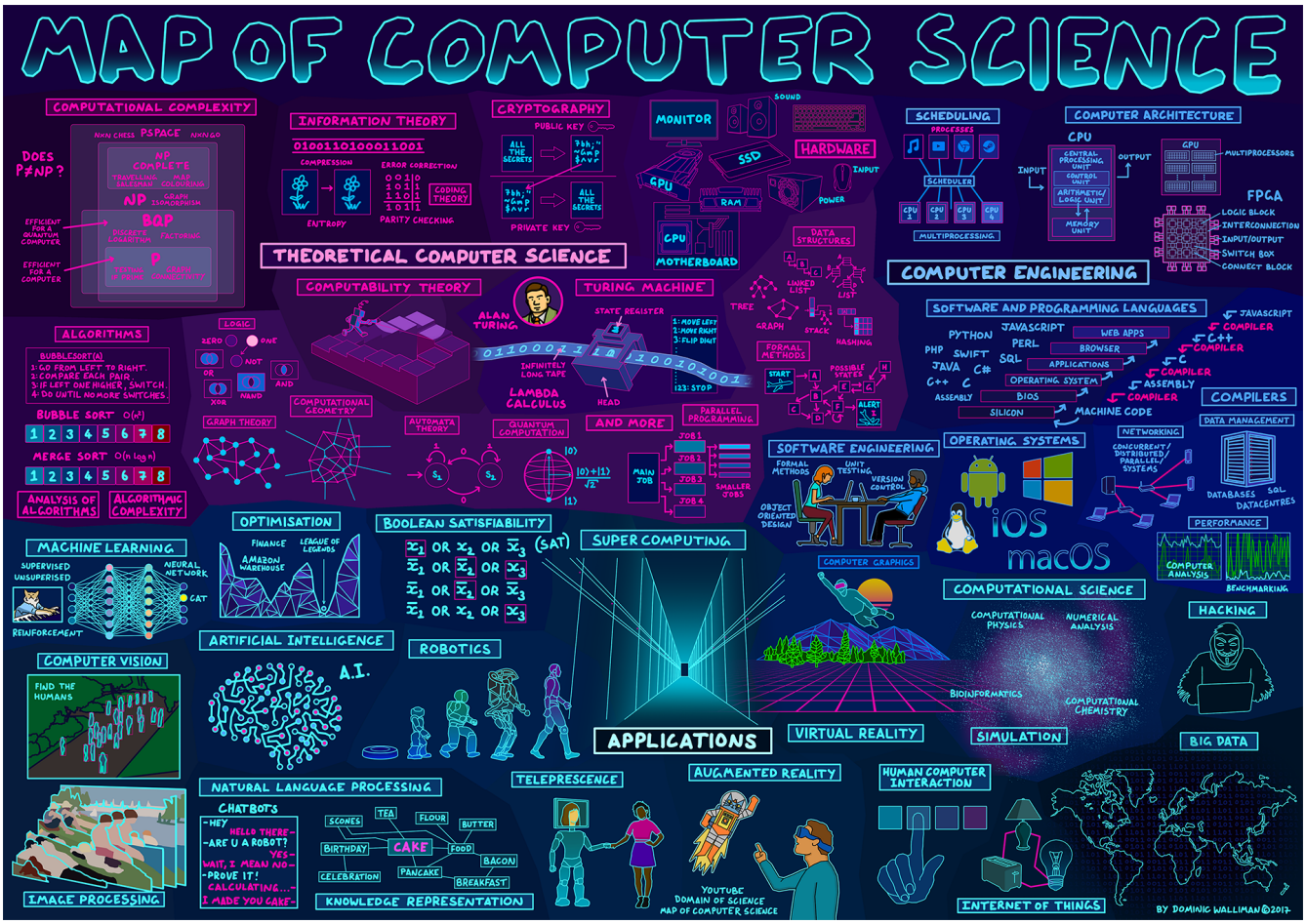
Interacción y Experiencia de Usuario

Contenidos



1. PRESENTACIÓN

1.1. Datos de la asignatura

Temario

1. **End-user development** y lenguajes visuales y de bloques
2. **Interacción multimodal** y verbal
3. **Entornos inmersivos**: realidad virtual, aumentada y mixta
4. **Videojuegos y aprendizaje**: ludificación y evaluación
5. **Usabilidad y experiencia de usuario**

Evaluación

- Evaluación continua del trabajo en la asignatura

Procedimiento

- Elaboración, entrega y defensa de un trabajo práctico para cada uno de los siguientes bloques a lo largo del curso
  - **End-user development**: Aplicación móvil con AppInventor
  - **Interacción verbal**: Agente conversacional y workflow con N8N
  - **Entornos inmersivos**: AR/VR/XR con Unity
  - **Videojuegos y aprendizaje**: Twine
  - **Usabilidad y experiencia de usuario**: Evaluación de *User eXperience* (UX)
- Calificación: **20%** entrega y defensa de cada bloque
- La defensa de algunos bloques podrá hacerse con la grabación de un video

Bibliografía

- Shneiderman, Plaisant, Cohen, Jacobs, Elmqvist (2018). *Designing the User Interface. Strategies for Effective Human-Computer Interaction*, 6th Edition, Pearson.
- A. J. Ko: *User Interface Software and Technology*, http://faculty.washington.edu/ajko/books/user-interface-software-and-technology/index.html
- Benyon, D. (2013): *Designing Interactive Systems. A comprehensive guide to HCI, UX and interaction design*, Addison Wesley.

1.2. Un poco de historia

Cronología

1940's	Vannevar Bush — Visionario del hipertexto e internet
1960's	Douglas Engelbart — Stanford SRI, Inventor del ratón
1970's	Alan Kay — Xerox PARC, WIMP
	• Diciembre 1979 ⇒ Steve Jobs visita el Xerox PARC
1983-1984	Apple Computer — Lisa + Macintosh
1985-1995	Bill Gates — Microsoft Windows 1.0 / 3.1 / 95
2000's	Steve Jobs (again)
	• 2007 ⇒ iPhone, pantalla táctil capacitiva
2017-2020	Deep learning, transformers
	• 2022 ⇒ Sam Altman — ChatGPT

Historia de las interfaces de usuario (WIMP)

> A user interface is the space where interactions between humans and machines occur
>
> https://history.user-interface.io/

Vannevar Bush

Fue un físico estadounidense que trabajó en el MIT y en el Laboratorio de Investigación de la Oficina de Servicios de Guerra durante la Segunda Guerra Mundial. En 1945 publicó un artículo en la revista *Atlantic Monthly* titulado *As We May Think* en el que propuso un sistema de almacenamiento y recuperación de información basado en la idea de un hipotético dispositivo denominado *memex*.

1940's: As We May Think

A SCIENTIST OF THE FUTURE RECORDS EXPERIMENTS WITH A TINY CAMERA FITTED WITH UNIVERSAL-FOCUS LENS. THE SMALL SQUARE IN THE EYEGLASS AT THE LEFT SIGHTS THE OBJECT

AS WE MAY THINK

A TOP U. S. SCIENTIST FORESEES A POSSIBLE FUTURE WORLD IN WHICH MAN-MADE MACHINES WILL START TO THINK

by VANNEVAR BUSH

DIRECTOR OF THE OFFICE OF SCIENTIFIC RESEARCH AND DEVELOPMENT

Condensed from the *Atlantic Monthly*, July 1945

This has not been a scientists' war; it has been a war in which all have had a part. The scientists, burying their old professional competition in the demand of a common cause, have shared greatly and learned much. It has been exhilarating to work in effective partnership. What are the scientists to do next?

For the biologists, and particularly for the medical scientists, there can be little indecision, for their war work has hardly required them to leave the old paths. Many indeed have been able to carry on their war research in their

ress, and the effort to bridge between disciplines is correspondingly superficial.

Professionally our methods of transmitting and reviewing the results of research are generations old and by now are totally inadequate for their purpose. If the aggregate time spent in writing scholarly works and in reading them could be evaluated, the ratio between these amounts of time might well be startling. Those who conscientiously attempt to keep abreast of cur-

Visionario del hipertexto y la Internet

Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, "memex" will do. A **memex is a device** in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an **enlarged intimate supplement to his memory**...

— Vannevar Bush, As We May Think

Memex fue el hipotético dispositivo hipertextual propuesto por Bush, un sistema de almacenamiento y recuperación de información que permitía la navegación entre documentos y la creación de hipervínculos entre ellos. Bush propuso que el dispositivo fuera capaz de almacenar y recuperar información de forma automática, de forma que el usuario pudiera concentrarse en la búsqueda de información y no en la gestión de la misma.

Memex animation [2'33'']

▶ https://www.youtube.com/watch?v=c539cK58ees *(YouTube video)*

Ivan Sutherland

En 1963, en el MIT, desarrolló el primer prototipo de un dispositivo de interacción humano-computador, el *Sketchpad*, que permitía al usuario interactuar con la computadora dibujando en una pantalla. Este dispositivo fue el antecedente de los sistemas de interacción gráfica de hoy en día.

1960's: Sketchpad

▶ https://www.youtube.com/watch?v=oNmJUcqISzQ *(YouTube video)*

Premio BBVA 2019 Fronteras del conocimiento en categoría TIC a Ivan Sutherland [2'35'']

▶ https://www.youtube.com/watch?v=6orsmFndx_o *(YouTube video)*

Steven A. Coons explicando Sketchpad [2'35'']

Douglas Engelbart

En 1962, en el Stanford Research Institute, desarrolló el *NLS* (*oN-Line System*), un sistema de interacción humano-computador que permitía al usuario interactuar con la computadora a través de un teclado, un ratón y una pantalla. Este sistema fue el antecedente de los sistemas de interacción gráfica de hoy en día.

Alan Kay

En 1968, en el Xerox Palo Alto Research Center (PARC), desarrolló *Smalltalk*, un lenguaje de programación orientado a objetos que permitía la creación de interfaces gráficas de usuario. Este lenguaje fue el antecedente de los lenguajes de programación orientados a objetos de hoy en día.

En 1973, Xerox desarrolló Xerox Alto, el primer prototipo con interfaces gráficas de usuario WIMP (Windows, Icons, Menus, Pointer).

WYSIWYG

▶ https://www.youtube.com/watch?v=J33pVRdxWbw *(YouTube video)*

Interfaz WIMP de Xerox Alto (1973) [2'15"]

▶ https://www.youtube.com/watch?v=54CyHBI1d6Y *(YouTube video)*

Xerox - The Company that threw away everything [30"]

La interfaz WIMP de Xerox Alto fue la primera GUI que permitía la creación de ventanas, iconos, menús y ratón. Esta interfaz fue el antecedente de las interfaces gráficas de usuario de hoy en día. Xerox Alto (1973) no fue un producto comercial, pero su tecnología fue licenciada primero por Apple y luego por Microsoft para sus propios sistemas operativos. Xerox Star (1981) fue el primer producto comercial que utilizó la interfaz WIMP, pero no tuvo éxito en el mercado debido a su precio.

WIMP

WIMP = Windows, Icons, Menus and Pointer

Steve Jobs

Steve Jobs visitó el PARC de Xerox en 1979 y se llevó consigo la idea de la interfaz gráfica de usuario WIMP.

En 1983, Jobs presentó el *Apple Lisa*, el primer ordenador personal de Apple que operaba con interfaz gráfica de usuario WIMP y un ratón. La tecnología del Lisa fue utilizada posteriormente en el *Apple Macintosh* (1984), que se convirtió en el primer ordenador personal de éxito comercial con interfaz gráfica de usuario WIMP.

▶ https://www.youtube.com/watch?v=J33pVRdxWbw *(YouTube video)*

How Steve Jobs got the ideas of GUI from XEROX [9'40"]

Interfaces WIMP

▶ https://www.youtube.com/watch?v=9qtd8Hc90Hw *(YouTube video)*

Interfaz WIMP del Apple Macintosh (1987) [1'21"]

Ver la película "Steve Jobs"

Bill Gates

Bill Gates incorpora la interfaz WIMP a Windows.

Bill Gates a Steve Jobs, sobre la tecnología de Xerox

> Creo que es más como si ambos tuviéramos este vecino rico llamado Xerox y yo irrumpí en su casa para robar el televisor y descubrí que tú

> [Steve Jobs] ya lo habías robado

Steve Jobs (again)

Apple incorpora las pantallas táctiles capacitivas a sus dispositivo móvil iPhone, lo que amplía el interés en nuevas formas de **entrada** y en las interfaces gráficas de usuario basadas en **gestos**.

Es importante distinguir entre las mejores interfaces de usuario para **entrada** y las mejores para **salida**.

Sam Altman

Sam Altman, director ejecutivo de Open AI, propone el término *Language User Interface* (LUI) para referirse a las interfaces de usuario basadas en **lenguaje natural**:

- Interacciones conversacionales (chatbots)
- Sugerencias generadas por *Large Language Models* (LLM)
- Interacciones contextualizadas

A pesar de que los LLM son una tecnología clave para la interacción *verbal* de **entrada** (chatbots), las interfaces de **salida** siguen siendo muchas veces gráficas, visuales o de otro tipo (por ejemplo, hápticas).

1.3. Un poco de teoría

Interfaces de usuario

> Software y hardware que salvan las diferencias entre la acciones humanas y las acciones del ordenador

- Comportamiento **funcional** determinista del ordenador
- Las UI (*User Interface*) hacen un **mapping** de las capacidades sensoriales, cognitivas y sociales del humano hacia las **funciones** de la máquina
- Las UI deben ofrecer representaciones "aprendibles" (*learnable*): algunas lo hacen y otras no

Learnability ("aprendibilidad")

> Facilidad con que un producto o aplicación de software puede ser comprendido y utilizado por los usuarios

¿Cuáles de las siguientes interfaces proporcionan representaciones *aprendibles*?

- ¿Un ratón?
- ¿Una pantalla táctil?
- ¿Una interfaz de voz?
- ¿Una interfaz de realidad virtual?

¿Todas las UI proporcionan un comportamiento funcional determinista?

- Ver el video Emotiv brain—computer interface (BCI) with portable EEG

Disciplinas involucradas

- Ciencia de la computación (en España, Informática)
- Sistemas de información (en España, Informática; fuera, *Business*)
- Ingeniería del software (en España, Informática)
- Inteligencia artificial (en España, Informática)
- Ergonomía y diseño de producto (en España, Diseño industrial y desarrollo de producto)
- Diseño gráfico
- Psicología cognitiva y Neuropsicología
- Sociología
- Lingüística [computacional]
- *Information Science* (en España, Biblioteconomía y documentación)
- Filosofía y Antropología

Múltiples teorías

- Teoría de la actividad
- Modelos conceptuales
- Procesos de diseño
- Búsqueda de información
- Semiótica
- Comunicación
- Etc.

- La teoría de la actividad se centra en el análisis de las actividades que el usuario realiza con el sistema y en la descripción de las tareas que el usuario realiza con el sistema.
- Los modelos conceptuales se basan en la idea de que el usuario tiene una representación

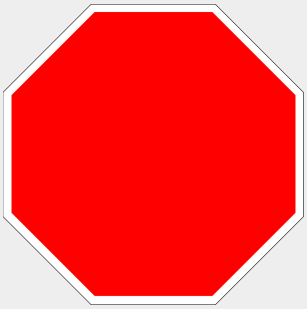
mental del sistema y que esta representación mental se construye a partir de la experiencia del usuario con el sistema.

- La semiótica se basa en la idea de que el usuario interpreta el sistema como un conjunto de signos.

Affordances ("asequibilidades") y **feedback** (retroalimentación)

- **Affordance** = propiedad de las interfaces que indica qué puede hacerse con ellas
  - Por ejemplo: la forma de un artefacto
  - En semiótica, affordance = el significante
  - Compuestos de hardware y software
- **Feedback** = las interfaces deben responder a las acciones del usuario
  - Por ejemplo, el sonido del click
- Las interfaces deben permitir construir un modelo mental de cuáles son las acciones posibles (el significado)

¿Qué implica este significante?



1.4. Temario detallado

Tema 1. Interacción + Lenguajes visuales y de bloques

Teoría	**Herramientas**
Estilos de interacción *End-user development* *Lenguajes visuales de bloques*	*Scratch, Snap!, App Inventor, Tinker CAD, Workflows N8N, etc.*

Tema 2. Interacción multimodal y verbal

Teoría	**Herramientas**
Interacción multi-modal *Interfaces verbales*	*DialogFlow, IBM Watson, Amazon Lex, Rasa NLU, N8N, ChatGPT, etc.*

Tema 3. Entornos inmersivos: realidad virtual, aumentada y mixta

Teoría	**Herramientas**
Conceptos *Dispositivos y demostración*	*Vuforia, Unity 3D, VEDILS, etc.*

Tema 4. Videojuegos y aprendizaje: evaluación y ludificación

Teoría	**Herramientas**
Videojuegos como herramientas de aprendizaje, gamificación y evaluación	*Twine, H5P, etc.*

Tema 5. Usabilidad y experiencia de usuario

Teoría	**Herramientas**
Usabilidad, Experiencia de usuario, Accesiblidad, Arquitectura de la información, Diseño de la interacción	*Evaluación de UX*

2. ESTILOS DE INTERACCIÓN

Estilos de interacción representativos

Manipulación directa
: Interacción mediante acciones físicas sobre objetos virtuales que simulan propiedades y comportamiento del mundo real.

Navegación y selección por menús
: Uso de listas o paneles jerárquicos para explorar opciones y elegir comandos o funciones.

Rellenado de formularios
: Introducción de datos mediante campos específicos para estructurar la información que el sistema procesa.

Lenguajes de comandos

Interacción textual donde el usuario escribe órdenes exactas para que el sistema las ejecute.

Lenguaje natural (humano)

Comunicación verbal mediante texto o voz usando lenguaje cotidiano y entendible sin codificación técnica.

Entornos inmersivos

Interfaces que sumergen al usuario en un espacio digital tridimensional, permitiendo interacciones realistas dentro del entorno virtual

---

2.1. Manipulación directa

- Es el paradigma inherente en las interfaces WIMP (Windows, Icons, Menus and Pointers)
- Uso intensivo del *drag-and-drop*
- Navegación fluida

Ejemplo 1. Manipulación directa para convertir bosquejos hechos a mano en gráficos vectoriales [1']

▶ https://www.youtube.com/watch?v=hI4w8J6yauk *(YouTube video)*

Características de la manipulación directa

1. Los objetos de interés siempre están representados con una metáfora **visual**
2. La forma de invocar comandos es interactuar con el objeto a través de **acciones físicas**
3. Se obtiene una **retroalimentación** inmediata de la acción sobre el objeto, que es **reversible**

Ejemplos de manipulación directa

- Sistemas de información geográfica con GPS
- Videojuegos
- CAD/CAM

¿Son las interfaces interactivas tipo WIMP naturales o aprendibles?

Ver la serie The Billion Dollar Code sobre los orígenes de Google Earth.

Futuro de la manipulación directa

¿Qué futuro espera a la manipulación directa?

- Pantallas táctiles
- Interfaces 2D y 3D
- Inteligencia artificial

Ejemplo 2. Conversión de imágenes bitmap en SVG usando ML [1'47"]

https://www.youtube.com/watch?v=4MfG9CDufPA *(YouTube video)*

Distancia traslacional

Distancia entre el usuario y la representación de la metáfora

- Débil
- Media
- Fuerte
- Inmersiva

¿Qué distancia traslacional suponen estas UI?

Distancias traslacionales

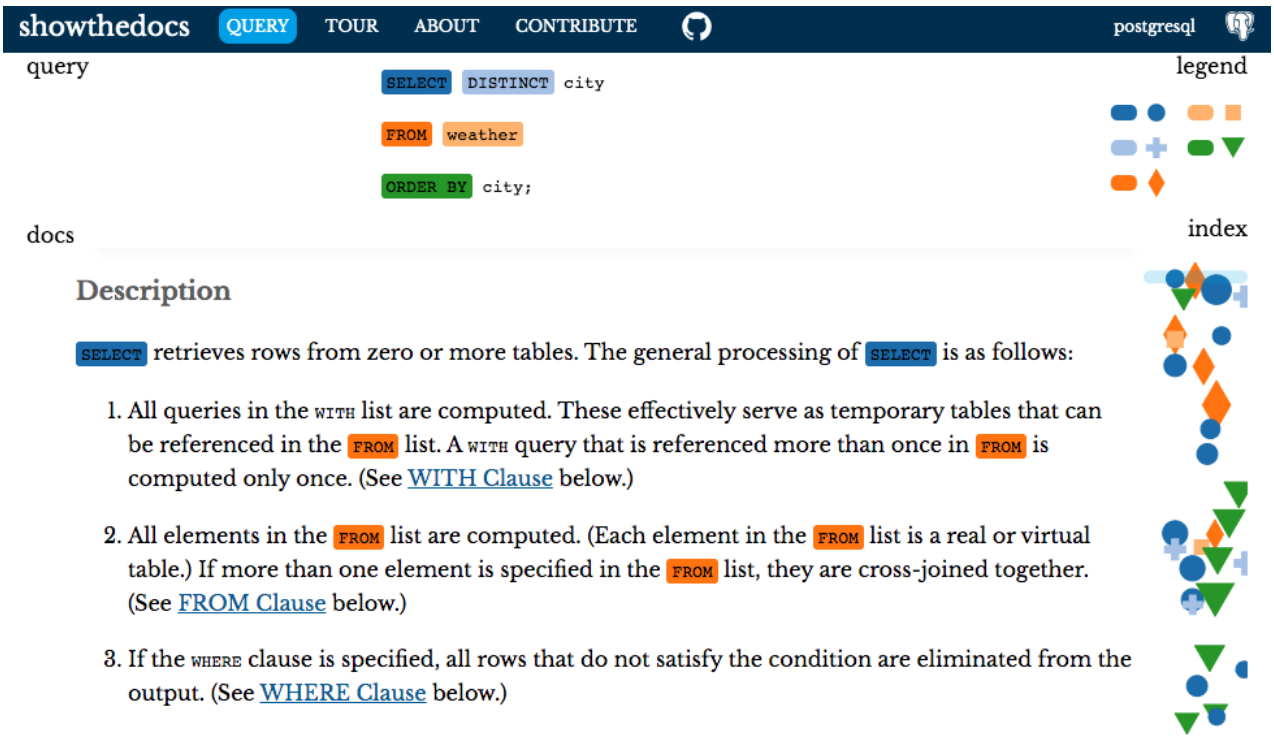
- **Débil** — e.g. mandos de videojuegos

- **Media** — e.g. pantallas táctiles y multi-táctiles
- **Fuerte** — e.g. guantes, gestos, tangibles
- **Inmersiva** — e.g. realidad virtual, realidad extendida

2.2. Lenguajes de comandos

Ejemplo 3. Lenguajes de comandos: shell

Ejemplo 4. Lenguaje de programación: SQL



Lenguajes para programación y configuración

- Macros y secuencias de acciones
- Lenguajes de scripting
- Lenguajes de configuración: v.g. terraform HCL
- Lenguajes de programación: v.g. terraform CDK
- Lenguajes de programación visual: v.g. Node-RED
- Lenguajes de bloques: v.g. Blockly

2.3. Lenguajes de programación visual

Los primeros lenguajes propuestos por el *End-user development*

- Lenguajes de bloques: *Block-based languages*
- Lenguajes de construcción y modelado: *Building Information Modeling* (BIM)
- Lenguajes mixtos: v.g. strype

Lenguajes de bloques

Lenguajes de bloques

- Blockly
- Scratch
- App Inventor
- Snap!
- MakeCode
- Tynker

Lenguajes de construcción y modelado

Lenguajes de construcción y modelado

- SysML
- Simulink
- BPMN
- Autodesk Revit

2.4. Lenguaje humano

A tratar en el tema *Interacción multimodal y verbal*

- Lenguaje natural
- Interfaces de voz

2.5. Entornos inmersivos

A tratar en el tema *Entornos inmersivos*

- Interacción gestual
- Realidad virtual
- Realidad aumentada

- Realidad mixta
- Realidad extendida

3. END-USER DEVELOPMENT

Los usuarios más avanzados suelen necesitar poder **personalizar** o ampliar las aplicaciones que manejan. Por ejemplo: Organizar recetas de cocina con Airtable.

- Algunos programas y apps pueden tener un cierto grado de libertad para configurar su comportamiento. Otros son cajas herméticamente cerradas a las que un usuario avanzado puede querer modificar de su comportamiento.
- Si no hay un botón que haga exactamente lo que el usuario quiera, su única alternativa es pedírselo al desarrollador del programa (al que le puede resultar conveniente hacerlo o no, y si lo hace normalmente querrá cobrar por ello).

Esta es una reclamación tradicional de los defensores del **software libre** y FLOSS (*Free and Libre Open Source*). En la práctica, reconstruir una aplicación y modificarla a partir del código fuente requiere un esfuerzo y exige habilidades avanzadas de desarrollo de software.

- Algunas herramientas permiten ***plugins*** que se ejecutan dentro de la aplicación principal para hacer las ampliaciones o modificaciones.
- Otras herramientas ofrecen una *Application Programming Interface* (**API**) para que los desarrolladores puedan hacer esas ampliaciones desde fuera y dejen a la aplicación principal hacer el resto.

El software debería ser ampliable y **extensible** de una forma más fácil y rápida, para que cualquier usuario final o *end-user* pueda automatizar, personalizar o construir su propia herramienta a partir de una dada. Para ello necesitan un lenguaje de *End-User Development* (**EUD**).

- El objetivo de la HCI/HMI pasa de hacer sistemas **fáciles de usar** a hacer sistemas **fáciles para desarrollar**
- Desafío ⇒ construir sistemas que permitan a usuarios sin conocimientos de programación **combinar** de forma creativa *artefactos* hechos de software
- Combinación de *artefactos* ⇒ crear, modificar y ampliar artefactos software
- Los *end-users* no son desarrolladores profesionales

Definición de EUD *(Lieberman et al. 2006)*

> A set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact
>
> — H. Lieberman et al., End-User Development: An Emerging Paradigm

Algunos ejemplos de EUD

- UNIX Pipes: For the love of pipes
- Hojas de cálculo:
  - Spreadsheets: A basis for end-user programming
  - Spreadsheets are code.
- La propia World Wide Web permite la creación de contenidos y aplicaciones personalizadas con HTML, CSS, Javascript, PHP, etc.
- Programación visual en ingeniería: LabView

3.1. End-User Programming

¿Por qué *end-user*?

Hace falta un término que distinga al desarrollo/programación por parte del usuario del desarrollo/programación hecho por profesionales de ingeniería de software

- Los ingenieros de software profesionales construyen aplicaciones de propósito general diseñadas para ser empleadas por muchas personas
- Los programadores end-user son usuarios avanzados, no necesariamente programadores profesionales, que modifican o crean herramientas *ad hoc* para su uso personal o para compartir con unos pocos colegas, todo lo más.

Lectura recomendada

- Ink & Switch (2019). End-user programming

¿Por qué algunos ingenieros prefieren Python y otros Excel?

- Calculation View: Representaciones múltiples de la edición en hojas de cálculo

Python en Excel

Microsoft ha incorporado Python en Excel en una versión preliminar para el programa *insider* del canal beta.

3.2. Computational Thinking

Los entornos y lenguajes de EUD ofrecen la capacidad de **crear** con una herramienta, más allá de la capacidad de **usar** una herramienta.

On Computing... *(Rosembloom 2013)*

> [...] incorporates both **understanding** (as is the focus in most traditional science) and **shaping** (as is the focus in engineering and other professions)
>
> — Paul Rosenbloom, On Computing: The Fourth Great Scientific Domain

Pero para poder crear y desarrollar software, hacen falta unos mínimos conocimientos y habilidades sobre los conceptos básicos sobre lo que un ordenador puede hacer, cómo lo hace y cómo instruirlo para que lo haga. Esto es el terreno del **pensamiento computacional** o *computational thinking* subyacente al *Computing* (en español, traducido por "Informática").

Computational thinking *(Wing 2006)*

> Computational Thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science.
>
> Computational thinking is a fundamental skill for everyone, not just computer scientist [...]; involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to CS [...]; is thinking recursively. It is parallel processing [...]; is using abstraction and decomposition when [...] designing a large complex system [...]; is thinking in terms of prevention, protection, and recovery from worst-case scenarios [...]; is using heuristic reasoning to discover a solution.
>
> — J. Wing, Computational Thinking -- Communications of the ACM

¿Recuerdan a Vannevar Bush?

A SCIENTIST OF THE FUTURE RECORDS EXPERIMENTS WITH A TINY CAMERA FITTED WITH UNIVERSAL-FOCUS LENS. THE SMALL SQUARE IN THE EYEGLASS AT THE LEFT SIGHTS THE OBJECT

AS WE MAY THINK

A TOP U. S. SCIENTIST FORESEES A POSSIBLE FUTURE WORLD IN WHICH MAN-MADE MACHINES WILL START TO THINK

by VANNEVAR BUSH

DIRECTOR OF THE OFFICE OF SCIENTIFIC RESEARCH AND DEVELOPMENT

Condensed from the *Atlantic Monthly*, July 1945

This has not been a scientists' war; it has been a war in which all have had a part. The scientists, burying their old professional competition in the demand of a common cause, have shared greatly and learned much. It has been exhilarating to work in effective partnership. What are the scientists to do next?

For the biologists, and particularly for the medical scientists, there can be little indecision, for their war work has hardly required them to leave the old paths. Many indeed have been able to carry on their war research in their

ress, and the effort to bridge between disciplines is correspondingly superficial.

Professionally our methods of transmitting and reviewing the results of research are generations old and by now are totally inadequate for their purpose. If the aggregate time spent in writing scholarly works and in reading them could be evaluated, the ratio between these amounts of time might well be startling. Those who conscientiously attempt to keep abreast of cur-

Visionario del *computational thinking*

> Wholly new forms of **encyclopedias** will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the memex and there **amplified**. The **lawyer** has at his touch the associated opinions and decisions of his whole experience, and of the experience of friends and authorities. The **patent attorney** has on call the millions of issued patents, with familiar trails to every point of his client's interest. The **physician**, puzzled by a patient's reactions, strikes the trail established in studying an earlier similar case, and runs rapidly through analogous case histories, with side references to the classics for the pertinent anatomy and histology. The **chemist**, struggling with the synthesis of an organic compound, has all the chemical literature before him in his laboratory, with trails following the analogies of compounds, and side trails to their physical and chemical behavior.
>
> — Vannevar Bush, As We May Think

Lectura recomendada

- Evan W. Patton, M. Tissenbaum & F. Harunani (2018). MIT App Inventor: Objectives, Design, and Development, en *Computational Thinking Education*, SpringerOpen,

DOI:10.1007/978-981-13-6528-7 (También disponible en MIT App Inventor.pdf)

3.3. *Low code*

El low code es una evolución del EUD, que permite a los usuarios crear aplicaciones sin necesidad de escribir mucho código.

Algunos ejemplos de lenguajes de low code son:

- Microsoft PowerApps
- Google AppSheet (sucesor de App Maker)
- Apple SwiftUI / Swift Playgrounds
- Oracle Visual Builder y Oracle APEX
- Salesforce Lightning
- Mendix
- GeneXus
- OutSystems

3.4. *Embodiment* o Encarnación

Una de las partes más difíciles de la construcción de software es que requiere que el programador mantenga varias abstracciones en su mente. El programador debe comprender el dominio del problema, modelar las estructuras de datos y el flujo del programa, y finalmente trasladar todo a una representación simbólica (**código** en un lenguaje).

La capacidad de razonamiento abstracto y concentración exigida a un ingeniero de software para escribir software no es viable para alguien que solo quiere resolver su problema particular mediante el ordenador y que suele ser experto en otros dominios ajenos al de la programación.

Hacen falta formas que requieran menos razonamiento abstracto y menos capacidad mental para modelar los programas y los datos, dejando más capacidad para pensar en el dominio de la aplicación.

Una forma de hacer esto es el *embodiment* o **encarnación**, es decir, ofrecer al usuario final representaciones concretas para los elementos de un programa que funcione.

Las encarnaciones son normalmente visuales, pero también de otros tipos (auditivas, hápticas, etc.)

Ejemplos de Encarnaciones

- Blockly: Encarnación gráfica (como bloques) en un lenguaje visual similar al usado por Scratch, App Inventor, etc.

- Seymour: Encarnación (como código) de la salida de un programa
- Bolt: Encarnación (gráfica) en un lenguaje visual de scripting para juegos en Unity

¿Excel usa alguna encarnación?

- Calculation View: **Encarnaciones** múltiples de la edición en hojas de cálculo

---

3.5. Programación literaria

- *Literate Programming*: Invento de Donald Knuth en 1983
- El código fuente se parece mucho más a un **documento** que a un listado de código
- Bloques de código embebido en un documento
- Genera un documento legible junto a código ejecutable

Ejemplos de programación literaria

- Jupyter: Entorno interactivo para crear *notebooks*, código y datos en una diversidad de lenguajes (Python, SQL, R, Matlab, etc.)

Lectura recomendada

- Justin Worthe: Literate programming

3.6. Herramientas

Herramientas GUI

- Adobe XD: Diseño y prototipado
- Balsamiq: Prototipado rápido
- Figma: Diseño y prototipado; freemium
- Lunacy: Diseño y prototipado; free
- Penpot: Diseño y prototipado; ¡open source!

Lenguajes block-based

- Blockly
  - Scratch
  - AppInventor
  - Snap!
  - Kodular
  - Beetle blocks
- Microsoft
  - MakeCode
  - MakeCode Arcade
- PencilCode
- Alice

Lenguajes data-flow

- Node-RED
- Lego Mindstorms V3

Lenguajes mixtos

- Strype

4. UX E INTELIGENCIA ARTIFICIAL

4.1. Evolución del EUD

El EUD ha evolucionado en tres grandes generaciones:

- En la primera, el usuario debía escribir **todo el código** directamente, lo que exigía conocimientos técnicos avanzados y limitaba la creación de soluciones a perfiles especializados.
- La segunda generación trajo plataformas de **low code** y **no code**, donde los usuarios podían construir aplicaciones y automatizaciones mediante interfaces visuales, arrastrando y configurando bloques, reduciendo la dependencia de programadores expertos.
- Una tercera generación, impulsada por la IA Generativa y el **prompting**, permite que los usuarios describan lo que quieren en lenguaje natural y el sistema genera, adapta o ejecuta el flujo automáticamente a través de **agentes inteligentes** (v.g. herramientas tipo n8n). Esto democratiza aún más la creación.

4.2. La IA y el futuro de las UI

The Weird Death of User Interfaces [14'33"]

▶ https://www.youtube.com/watch?v=KG4ONHqF1qg *(YouTube video)*

Según Enrico Tartarotti, la muerte de las UI que se han utilizado durante los últimos 40 años se considera un hecho. A pesar de los esfuerzos recientes por perfeccionar y embellecer estas interfaces, las mismas compañías están intentando eliminarlas por completo, reemplazándolas con simples "prompts de texto en blanco y negro" que recuerdan a las líneas de comando de los años 80.

Esta transición se debe a que, si bien las UIs tradicionales son muy buenas para la **salida** (mostrar información de forma visual), son deficientes en la **entrada** (la necesidad de navegar por menús y hacer clics).

Por el contrario, los nuevos chatbots han resuelto el problema de la entrada al "entender" el lenguaje natural, aunque su salida sigue siendo el texto plano. En el futuro post-UI, las interfaces no se construirán, sino que se generarán sobre la marcha a través de agentes que responden al lenguaje, lo que puede llevar a que las UIs desaparezcan en gran medida.

4.3. La IA como sistema operativo

OpenAI just made your entire tech stack obsolete... [15"]

▶ https://www.youtube.com/watch?v=91aH8jsG4cc *(YouTube video)*

El intento de OpenAI de usar ChatGPT como un sistema operativo (SO) o interfaz universal para todas las aplicaciones se resume en su ambición de convertir a ChatGPT en una plataforma de aplicaciones, buscando reemplazar o evitar la necesidad de las GUI tradicionales. Este esfuerzo ha pasado por varias fases y se basa en la idea de que la forma de interactuar con el software está cambiando fundamentalmente.

La analogía que se hace es que los LLM tienen propiedades que los asemejan a los sistemas operativos (SO).

- **ChatGPT como plataforma**: OpenAI está apostando fuertemente por convertir a ChatGPT en una plataforma de aplicaciones. El objetivo es que ChatGPT actúe como un SO que te mantenga completamente *encerrado* en su ecosistema.
- **La analogía del SO**: Los LLMs se ven como una especie de nuevo sistema operativo. El LLM en sí es como la CPU y las ventanas de contexto son como la memoria. El LLM orquesta computación y memoria para resolver problemas.

El experimento de la interfaz universal

OpenAI just made your entire tech stack obsolete... [30"]

▶ https://www.youtube.com/watch?v=91aH8jsG4cc *(YouTube video)*

OpenAI ha intentado varias aproximaciones para lograr que ChatGPT sea la interfaz de interacción principal:

- Inicialmente, lanzaron GPT Plugins
- Luego, intentaron una ligera variación llamada GPTs
- Actualmente, están intentando implementar *Apps in ChatGPT*.

El paradigma de la entrada y salida

La clave de este intento radica en la capacidad de los chatbots para manejar la entrada del usuario de forma humana, aunque su salida sea actualmente deficiente:

- **Entrada**: con un chatbot puedes "divagar" en modo de voz o escribir un prompt mal formado, y el LLM es muy bueno para entender lo que quieres.
- **Salida**: sin embargo, fallan en la salida; son malos mostrando la información o haciéndote sentir en control, volviendo a interfaces similares a las líneas de comando de los años 80.

Agentes autónomos

OpenAI just made your entire tech stack obsolete... [1'30"]

▶ https://www.youtube.com/watch?v=91aH8jsG4cc *(YouTube video)*

AgentKit de OpenAI es un conjunto unificado de herramientas diseñado para que los desarrolladores puedan construir, desplegar y optimizar agentes de IA (agents).

Es la alternativa de OpenAI a soluciones open source como n8n.

4.4. Autonomía parcial

Aplicaciones parciamente autónomas [21']

▶ https://www.youtube.com/watch?v=LCEmiRjPEtQ *(YouTube video)*

Andrej Karpathy considera las **aplicaciones parcialmente autónomas** como una de las mayores oportunidades en la era del software impulsado por LLM. Estas aplicaciones representan una forma de interactuar con los modelos de lenguaje que va más allá de simplemente usar el LLM directamente como una *terminal* o un *sistema operativo* (como copiar y pegar código en un chat).

Estas aplicaciones están diseñadas para la cooperación entre humanos e IA (*Human in the loop*). En

esta cooperación, la IA se encarga de la generación del trabajo, mientras que los humanos realizan la verificación o auditoría. El objetivo principal es hacer que este ciclo de generación y verificación sea lo más rápido posible.

Aplicaciones parcialmente autónomas

Las aplicaciones parcialmente autónomas, como Cursor (para programación) y Perplexity (para investigación), comparten varias propiedades fundamentales:

1. **Gestión del contexto**: Los LLM dentro de estas aplicaciones manejan una enorme cantidad de la gestión del contexto por el usuario.
2. **Orquestación de LLMs**: Orquestan múltiples llamadas a diferentes modelos LLM, incluyendo modelos de embeddings, modelos de chat y modelos específicos que aplican cambios (como diffs de código).
3. **GUI Específica**: Poseen una GUI especializada para la aplicación.
    - La GUI es vital para la auditoría, ya que utiliza la capacidad de *visión artificial* del cerebro humano.
    - Leer texto es un esfuerzo, pero mirar representaciones visuales es rápido y fácil. Por ejemplo, es más fácil ver un *diff* de código en rojo y verde que leerlo en texto, y es más rápido aceptar o rechazar un cambio con un comando de teclado que escribirlo.
4. **Control deslizante de autonomía** (*autonomy slider*): El usuario debe tener control sobre el nivel de autonomía que desea otorgar a la herramienta.
    - Esta autonomía se puede ajustar según la complejidad de la tarea.
    - El rango puede ir desde una baja autonomía (como la finalización con un toque o modificar solo un fragmento de código) hasta la autonomía total del agente (como permitir que el agente modifique todo un repositorio).

Atar en corto a la IA

Dado que los LLMs son sistemas falibles y propensos a alucinaciones y déficits cognitivos, es crucial "atar en corto" a la IA.

- La IA puede ser sobre-reactiva y generar resultados excesivamente grandes (como un diff de 10,000 líneas de código) que se convierten en un cuello de botella para la verificación humana.
- Para que el bucle de generación-verificación sea rápido, es mejor trabajar en **trozos pequeños e incrementales**. Esto se logra siendo más concreto en las indicaciones (prompts).

Al igual que sucedió con el piloto automático en la conducción autónoma (un producto de autonomía parcial), la dirección correcta es construir productos de autonomía parcial en lugar de grandes demostraciones llamativas de agentes totalmente autónomos. Durante la próxima década, la oportunidad reside en desarrollar software que mueva gradualmente ese control deslizante de autonomía hacia una mayor automatización.