UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 08

Usability and Human-Centered Design

Prof. Luigi Libero Lucio Starace

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

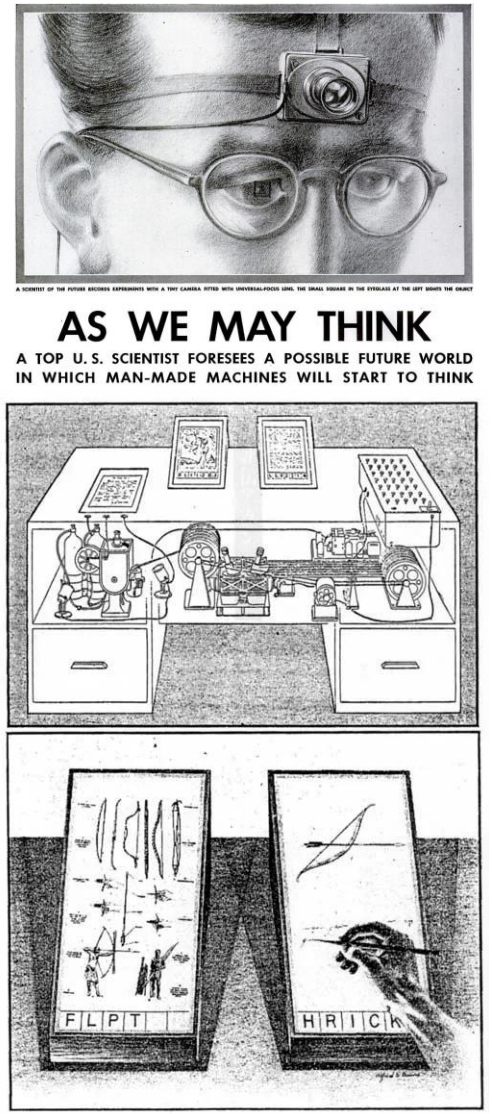
Historical Perspective: 1940s – 1960s

- The first computers were tools for calculation and symbolic manipulation
- One batch task at a time
- No «interaction» between computer and operator after starting a run
- Input was given via punch cards, tapes and plugs
- It was enough that it worked!

Patsy Simmers, holding ENIAC board; Gail Taylor, holding EDVAC board; Milly Beck, holding ORDVAC board; and Norma Stec, holding BRLESC-I board. U.S. Army/ARL Technical Library Archives

Historical Perspective: 1945

- Vannevar Bush published his essay «As We May Think»
- Postulated **Memex** device
  - Stores all records / articles / communications
  - Items retrieved by indexing, keywords, cross references (now called hyperlinks)
  - **Interactive** and **non-linear components** are key
- Computers were seen not only as tools for calculation, but as a way to **augment human intelligence**

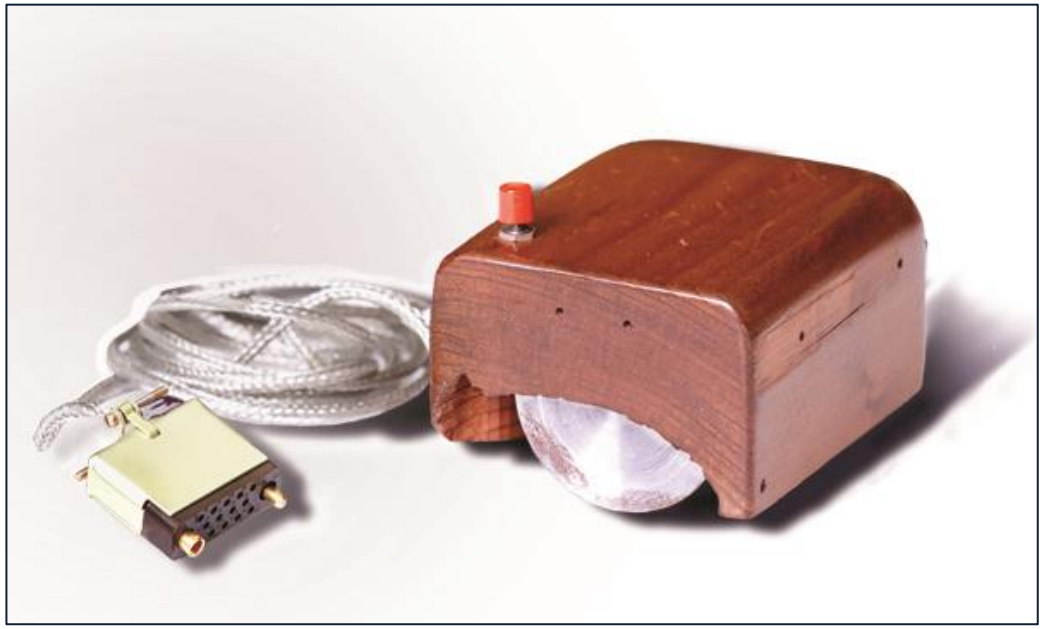


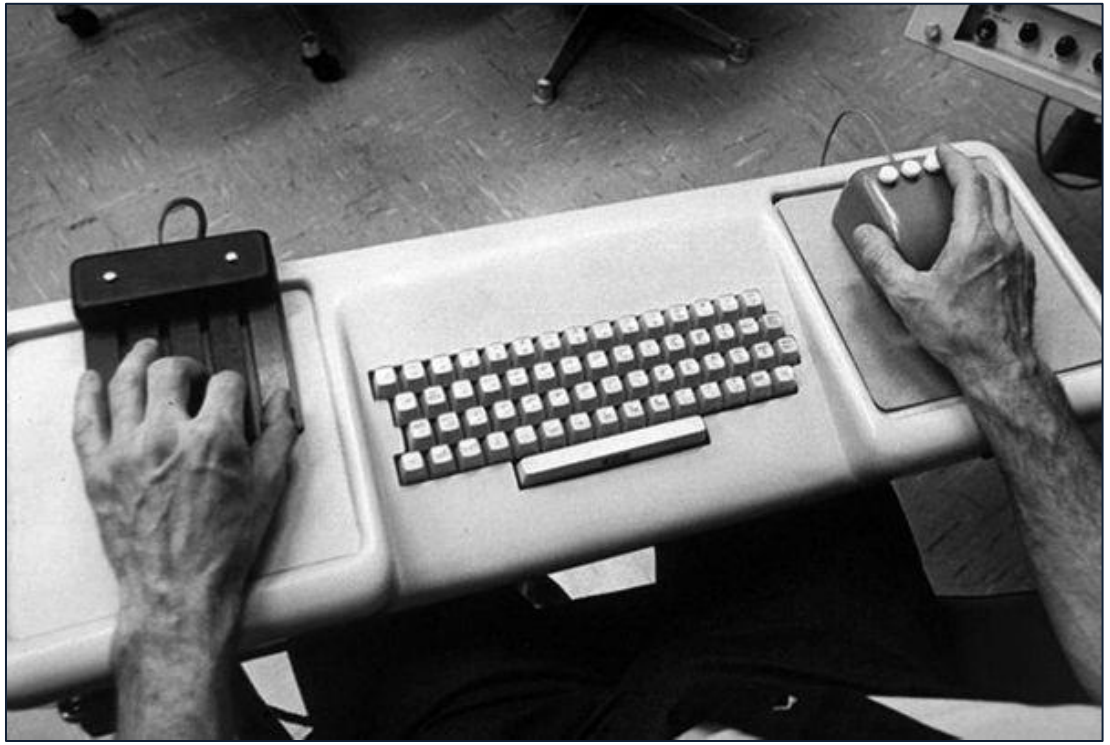
Historical Perspective: 1963

- Ivan Sutherland developed the first software with a Graphical User Interface (and won the Turing Award for this, in 1988!)

Historical Perspective: 1968

- Douglas Engelbart invented first computer mouse





Historical Perspective: 1970s - 1980s

- 1974 IBM 5100
- 1981 Datamaster
- 1981 IBM XT/AT
- 1984 Olivetti M21

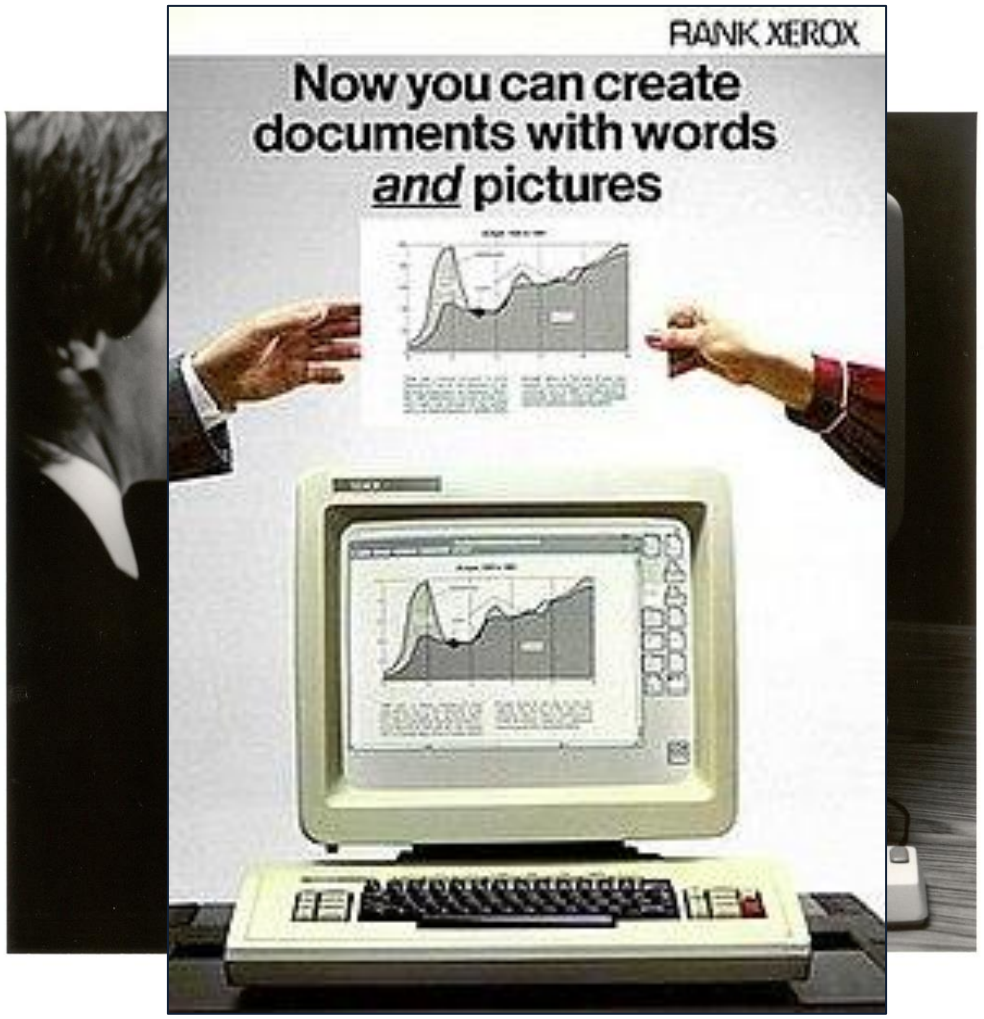
- Text and command-based
- Performed lots of tasks the general public wanted done
- A good basic toolkit

Historical Perspective: WIMP

- **WIMP** stands for **W**indows, **I**cons, **M**enus, **P**ointers
- Multitasking systems: WIMP interface allows you to do several things simultaneously
- Has become the familiar GUI interface we know and use to day
- First systems with WIMP interfaces started appearing in the second half of the 1970s and in the first half of the 1980s

Historical Perspective: Xerox Star (1981)

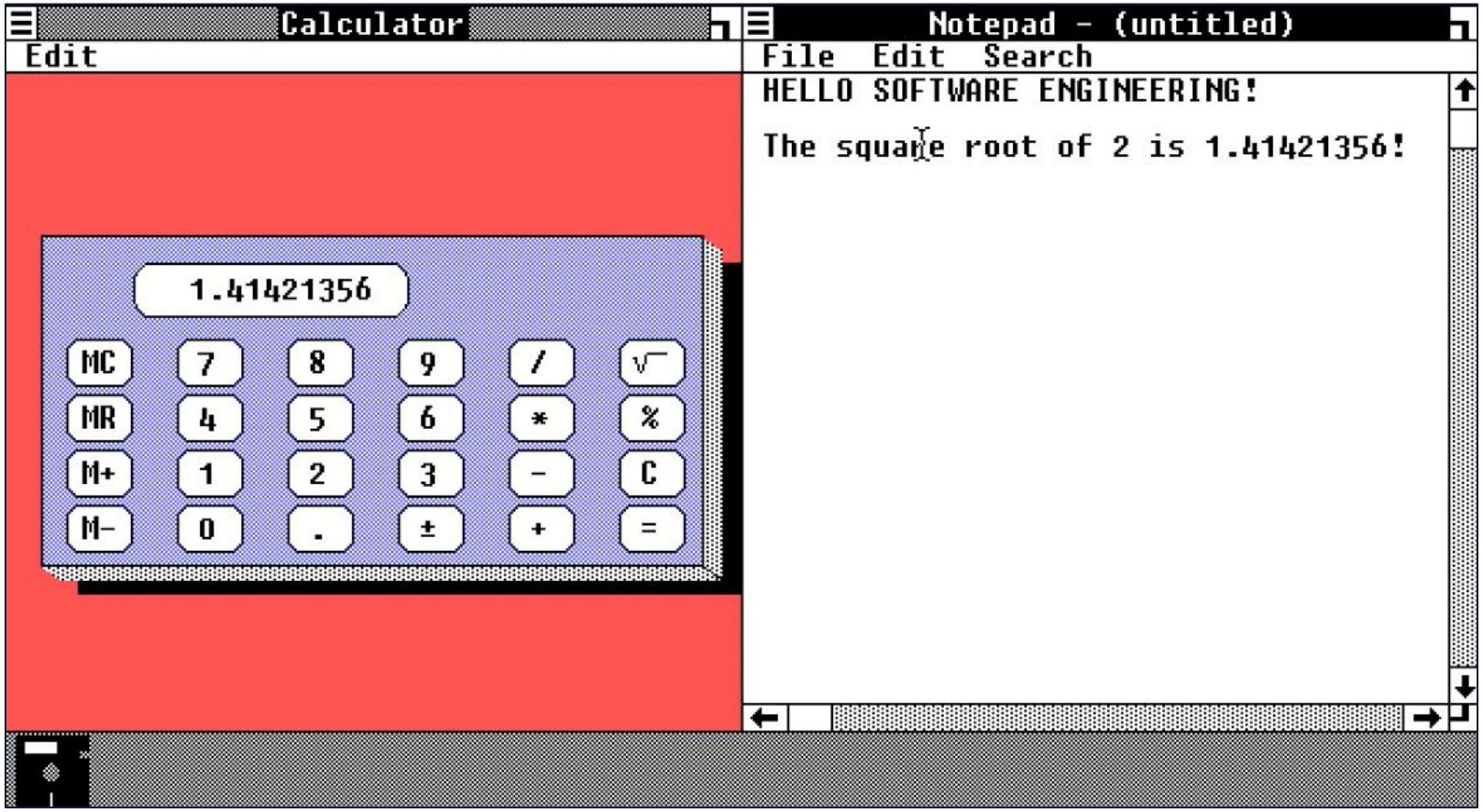
- First commercial PC designed for “business professionals”
  - desktop metaphor, pointing, WYSIWYG, consistency and simplicity
- First system based on usability
  - Paper prototyping and analysis
  - Usability testing & iterative refinement
- Commercial flop
  - $15k cost
  - closed architecture
  - lacking key functionality (spreadsheet)



Historical Perspective: Apple Macintosh (1984)

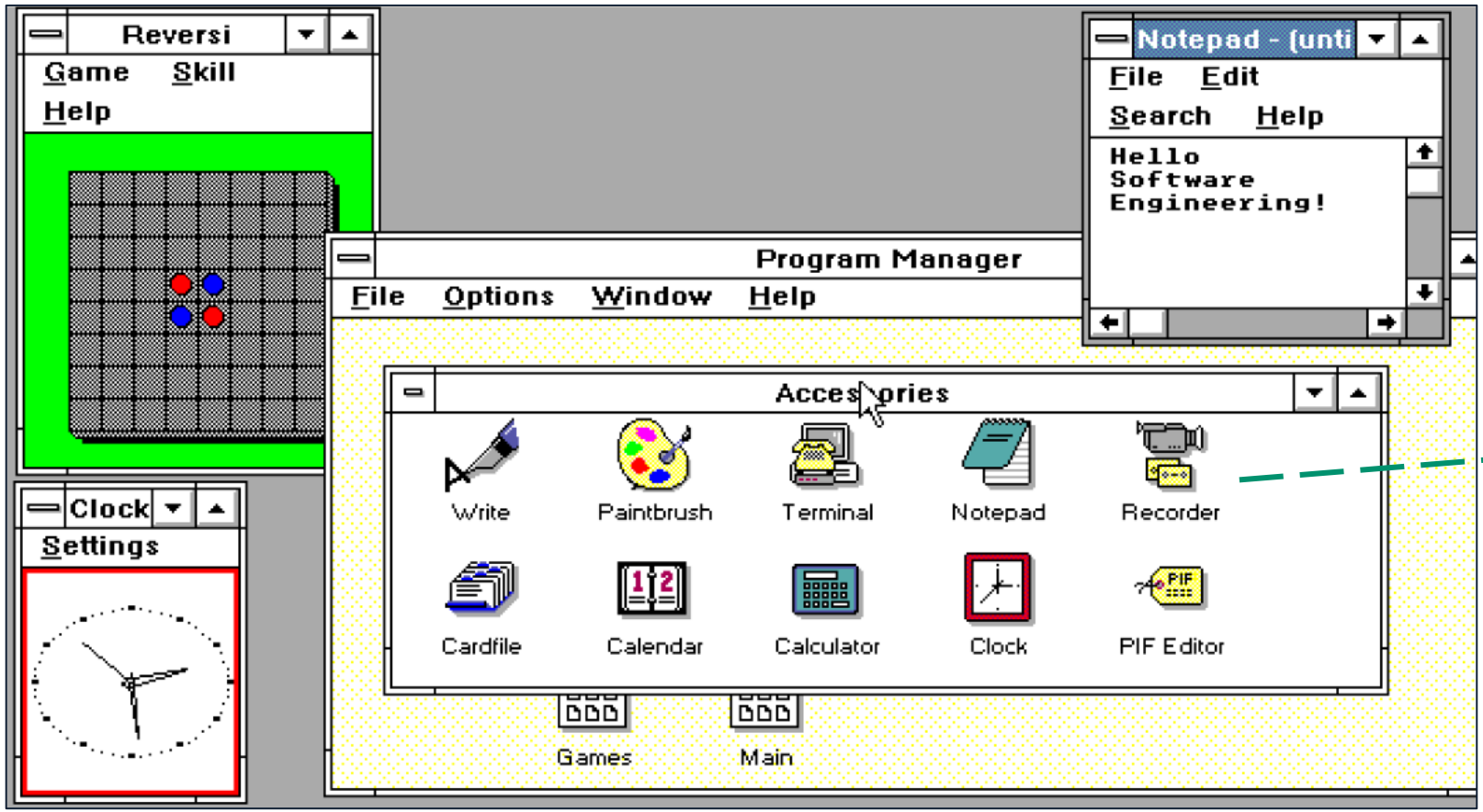
- Aggressive pricing: $2500
- 3rd party applications
- Good **interface guidelines**
- High quality graphics and laser printer

Historical Perspective: Windows 1.0 (1985)



Windows 1.0 – Emulated in a modern web browser at https://www.pcjs.org/software/pcx86/sys/windows/1.00/

Historical Perspective: Windows 3.0 (1990)

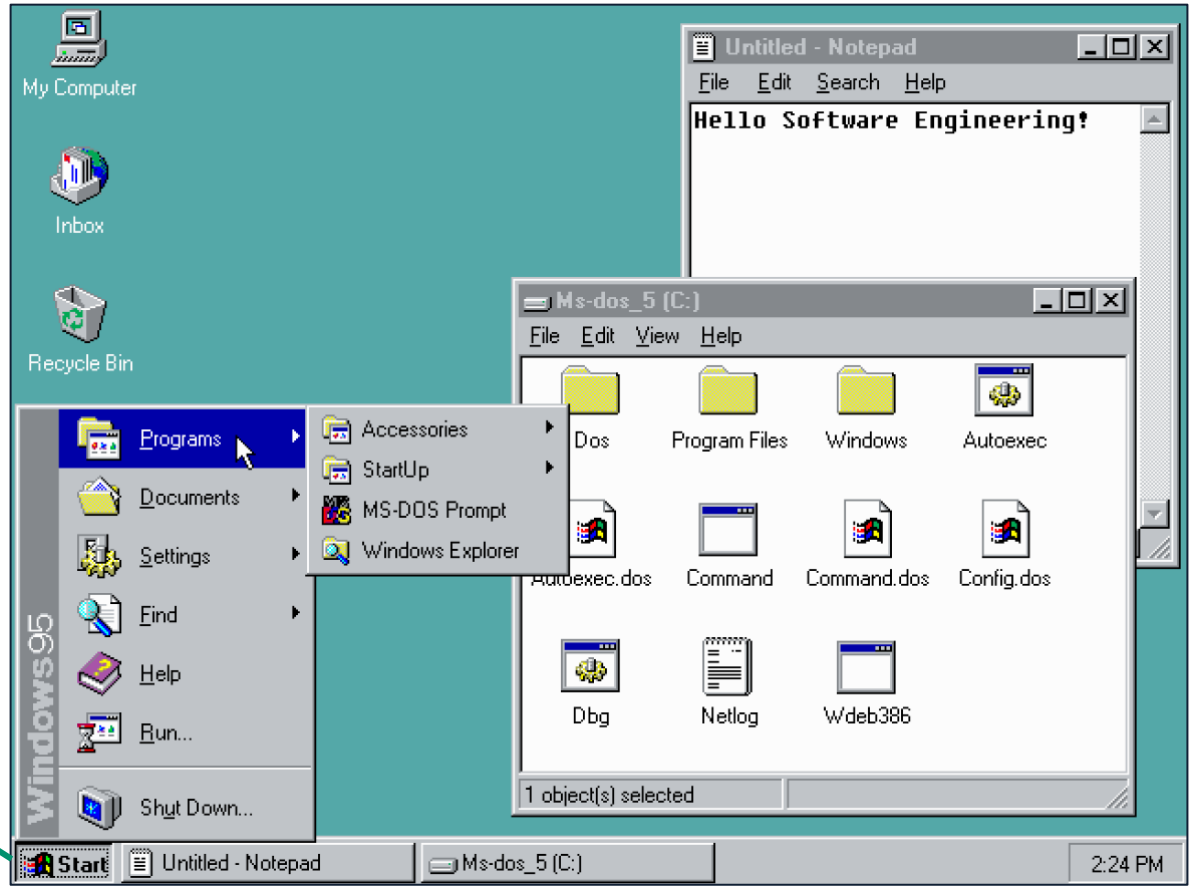


Programs represented as icons!

Windows 3.0 – Emulated in a modern web browser at https://www.pcjs.org/software/pcx86/sys/windows/3.00/

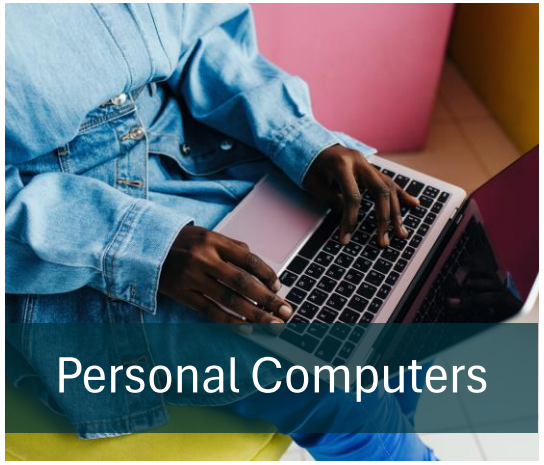
Historical Perspective: Windows 95 (1995)

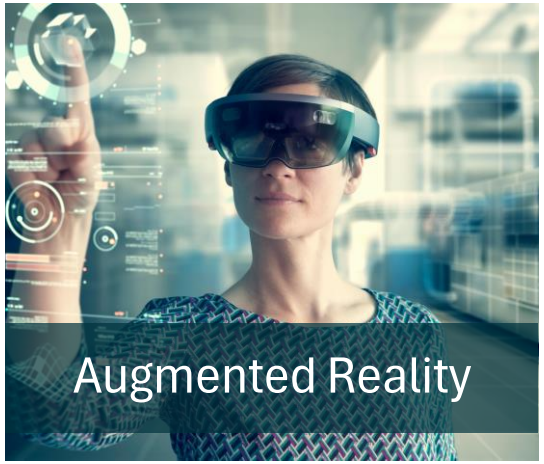
Start menu and taskbar introduced

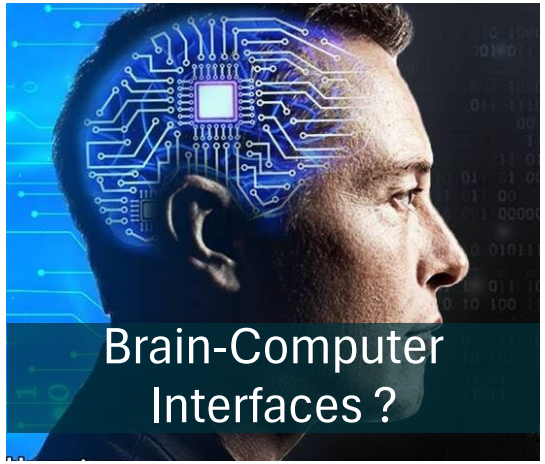


Windows 95 – Emulated in a modern web browser at https://www.pcjs.org/software/pcx86/sys/windows/win95/4.00.950/

Human-Computer Interaction: current trends



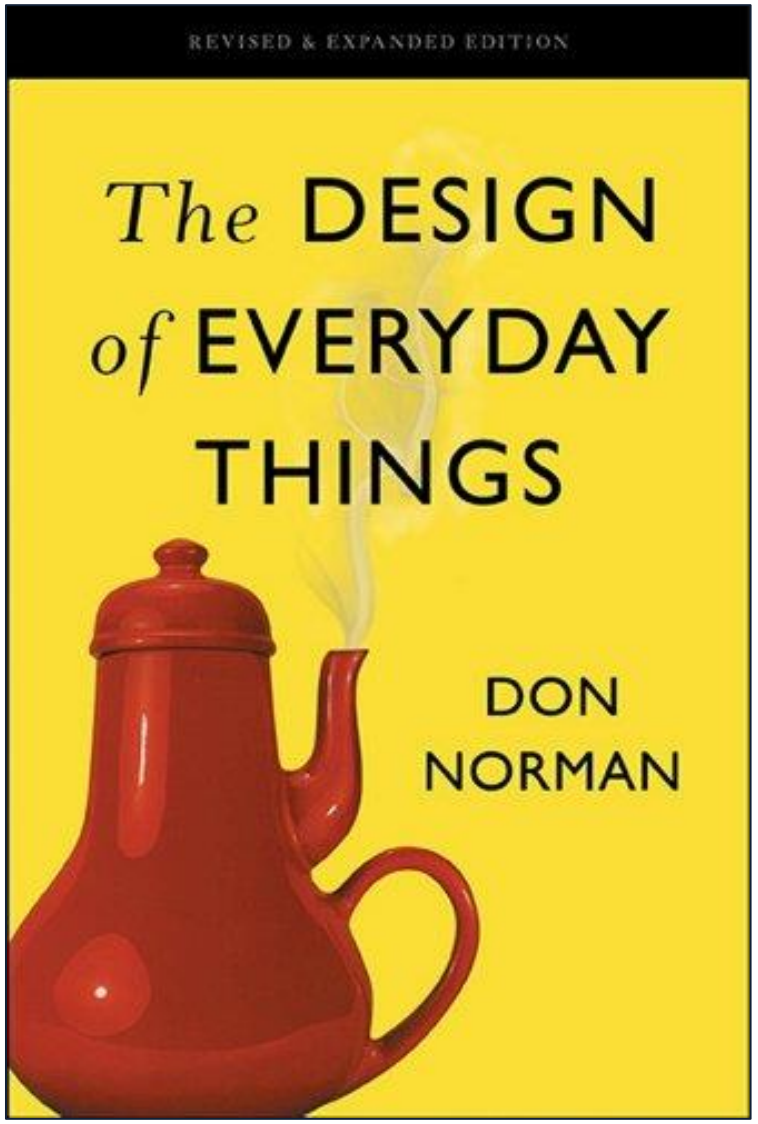




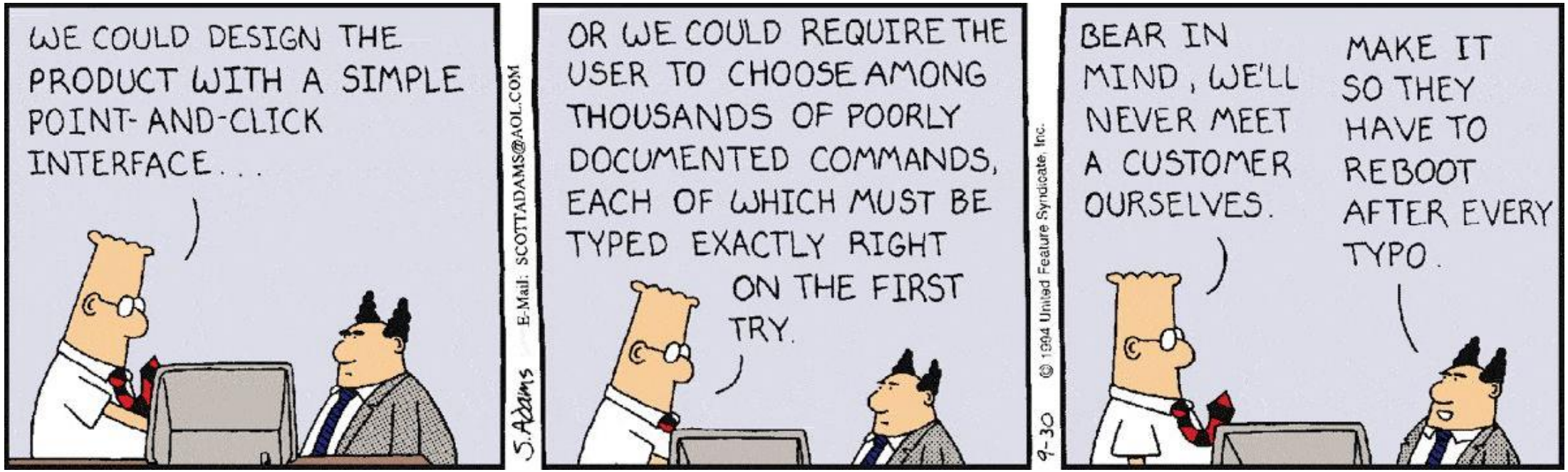
The Rise of Usability

- In the first decades of the computing era, usability was not a dominant concern among those who built software
- Don Norman, in The Design of Everyday Things (1988):

  To make technology that fits human beings, it is necessary to study human beings. But now we tend to study only the technology. As a result, people are required to conform to technology. It is time to reverse this trend, time to make technology that conforms to people.



The Rise of Usability



The Rise of Usability

- Back in the days, it was enough if the system just worked
- Back in the days, the user had to adapt to the product
  - Selling training and customer support could be an additional way to profit, there was little motivation to ship good user interfaces

The Rise of Usability

Today, we **must care** about building systems with a ***good usability***

- **Usability**: a qualitative measure of the **ease** and **efficiency** with which a human can employ the functions and features offered by the system.

Regardless of the type of software we are building, good usability can make the difference between success and failure, and even between life and death!

Usability - Apps for General Public

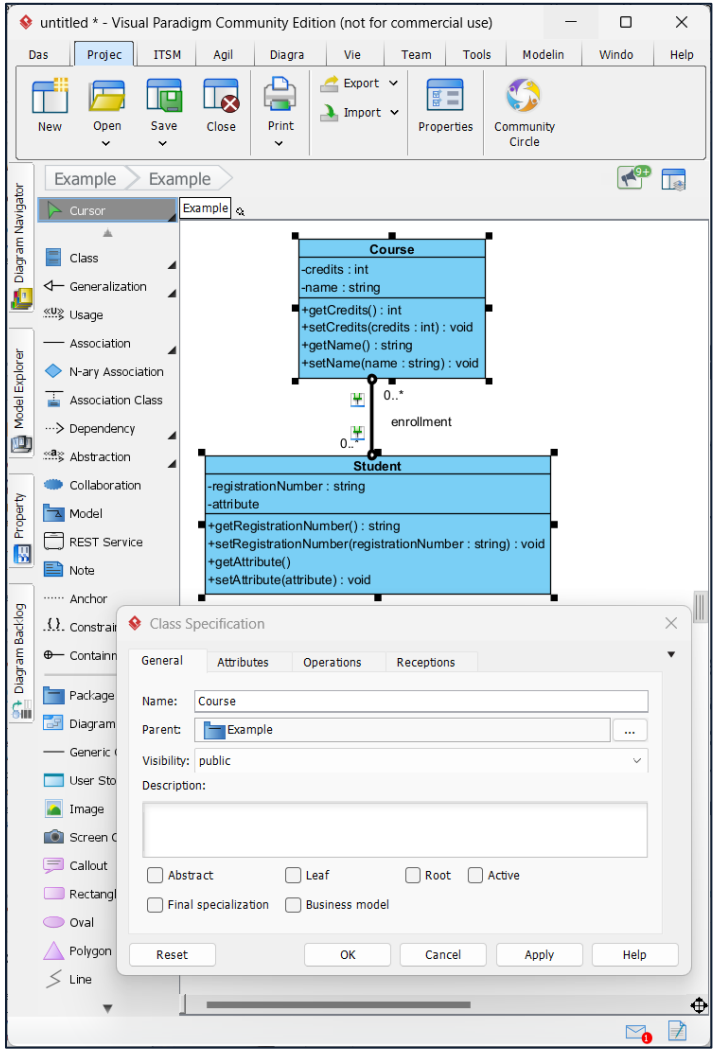
If we're building a new social media app, an e-commerce website, or yet another smartphone app for doing something:

- **ease of learning**, **low error rates**, and **subjective satisfaction** are paramount
- Use is discretionary and (generally) competition is fierce
- If the users cannot succeed quickly and efforlessly, they will give up and try a competing supplier

Usability – Professional Software

For software used in a professional environment (banking, insurance, production management, bookings and reservations, utility billing, ...)

- Training time is a cost, ease of use is important
- Internationalization may necessary
- Speed of performance is important, and operator fatigue, stress and burnout are concerns
  - Trimming 10% ot the mean transaction time could mean 10% fewer operators, 10% fewer workstations, ...



Usability – Life-critical Systems

For life-critical systems such as those that control air traffic, nuclear reactors, power utilities, emergency service dispatch, military operations, and clinical care)

- High costs due to lengthy training times are expected, but should yield rapid, error-free performance even when the users are under pressure
- Errors or delays in execution may cause serious damage!

Human-Computer Interaction (HCI)

HCI is the discipline that studies how humans interact with computers, and how to design **effective** and **usable** user interfaces

Human Abilities

(perception, memory, cognitive processes...)

Technologies

(VR, AR, mouse, windowing, touch screens, ...)

Methodologies

(prototyping, lifecycles, ...)

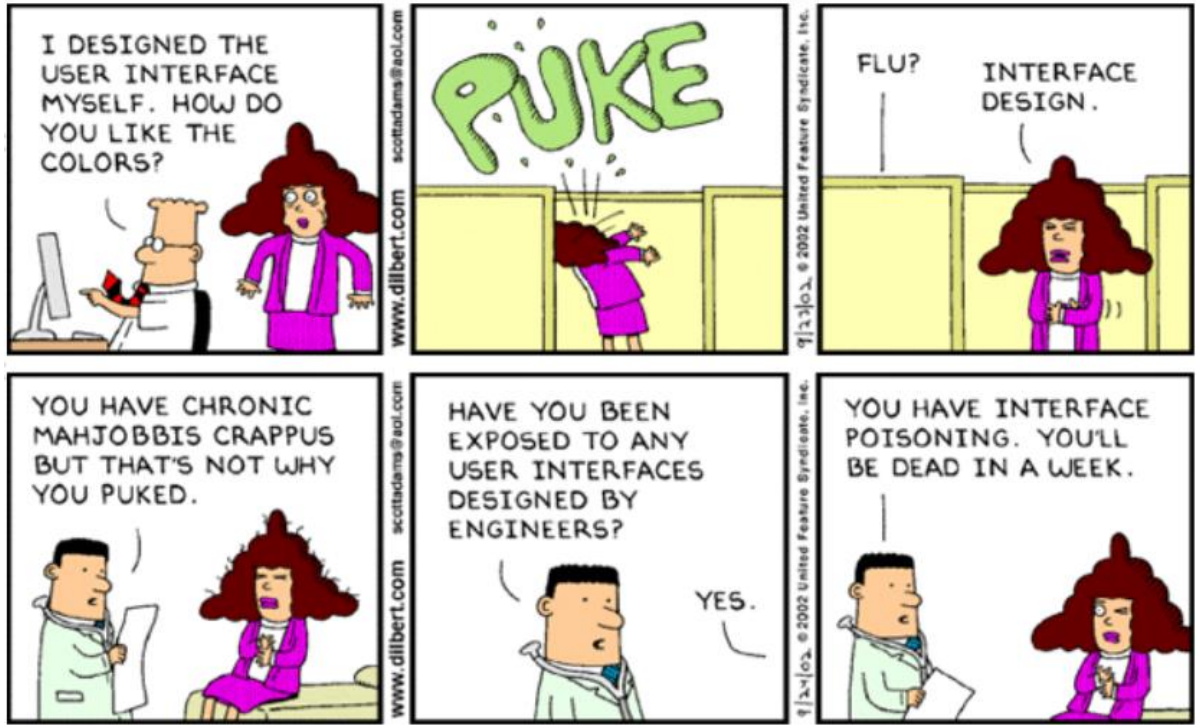
Guidelines, Principles and Theories

Evaluation

(experiments, observation, ...)

How do we build Usable UIs?

- Common sense?
- The road to horrible UIs is paved with the common sense of engineers!



Usability vs User-friendliness

When computer and software vendors started seeing users as more than an inconvenience, they started describing their systems as **user friendly**

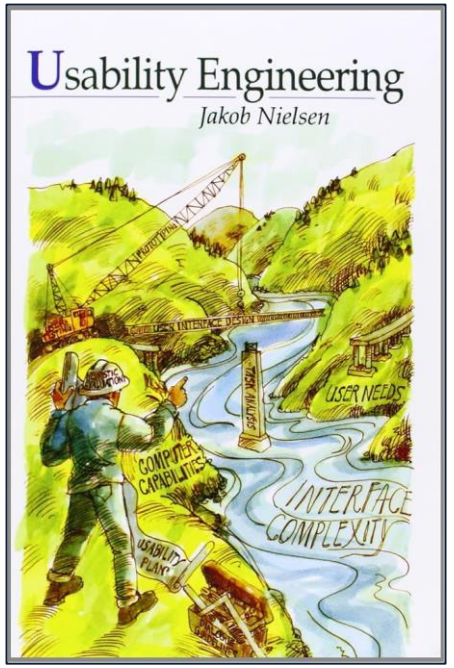
Not a very good term to use

- **Unnecessarily anthropomorfic:** users need systems that help them get their work done and do not stand in their way. They do not need systems to be friendly with them!
- **It implies that users' needs can be described along a single dimension** by systems that are more or less friendly
  - In practice, systems that are friendly to one user may feel tedious to others

Usability

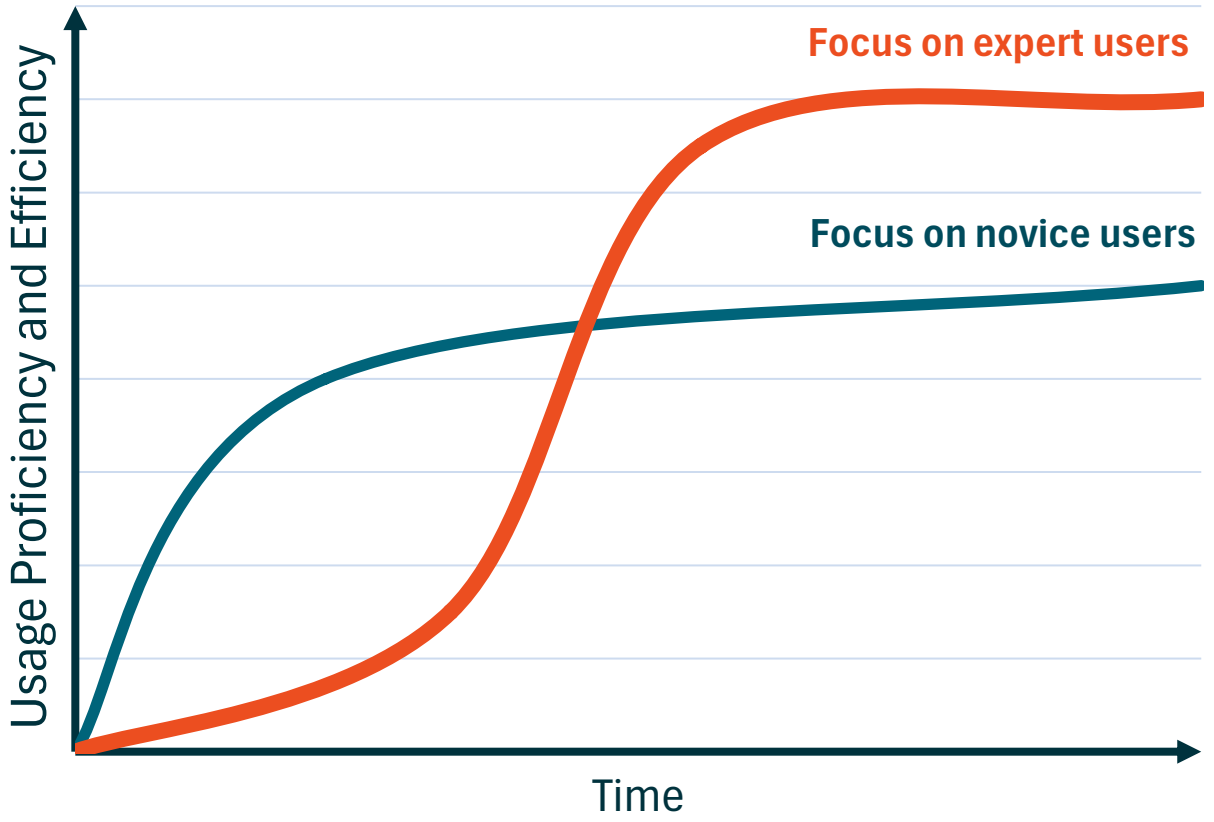
Jakob Nielsen defines it by means of **5 quality attributes**:

- **Learnability**: How easy is it for users to accomplish basic tasks the first time they encounter the design?
- **Efficiency**: Once users have learned the design, how quickly can they perform tasks?
- **Memorability**: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
- **Errors**: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- **Satisfaction**: How pleasant is it to use the design?



Learnability

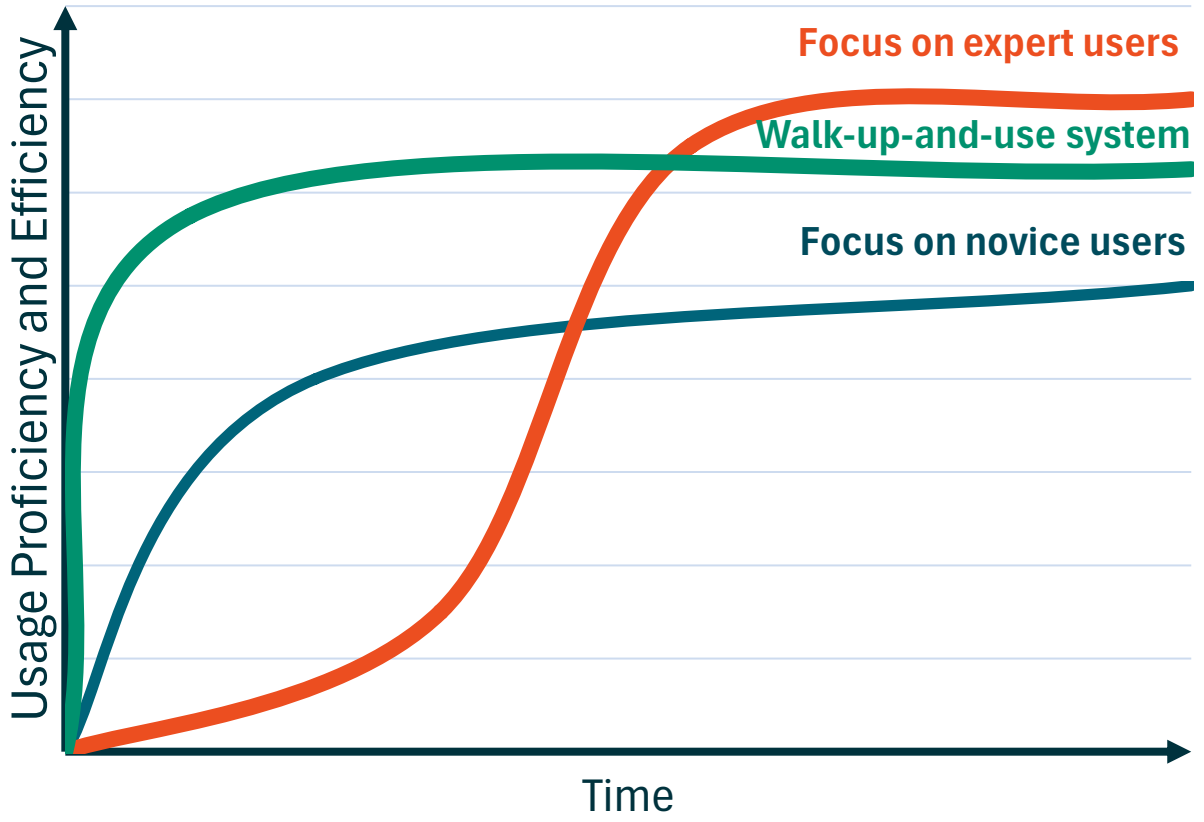
- Probably the most fundamental usability attribute
- Most systems need to be easy to learn
- For some specialized systems, being hard to learn but highly efficient for expert users is acceptable



Learning curves for different kinds of systems

Learnability

- So-called **walk-up-and-use** systems (e.g.: museum information systems) are only meant to be used once
- These systems need essentially zero learning time: users should be successful the first time they use them!



Learning curves for different kinds of systems

Efficiency of Use

- Efficiency refers to the expert user's steady-state level of performance at the time when the learning curve flattens out
  - It may take months or years to reach that stage!
- To measure efficiency:
  - Decide on some definition of expertise
  - Get a representative sample of users with that expertise level
  - Measure the time it takes them to complete some typical tasks

Memorability

- Casual users are a third category of users besides novices and experts
- Casual users use the system intermittently
- In contrast to novices, they do not need to learn it from scratch, but they need to remember how to use it based on their previous learning
- Casual use typically happens for:
  - Software that are not part of a user's primary work
  - Software that are inherently used at long intervals (e.g.: to draft yearly reports)
  - Software that are used only in exceptional circumstances
- Memorable interfaces are also useful for users who return to use the system after being on vacation, or have temporarily stopped using it

Memorability

- Improvements in learnability make an interface also easy to remember
- In principle, however, the usability of returning to a system is different from that of facing it for the first time
- What does this sign mean?
  - Not very learnable, but memorable!

Errors

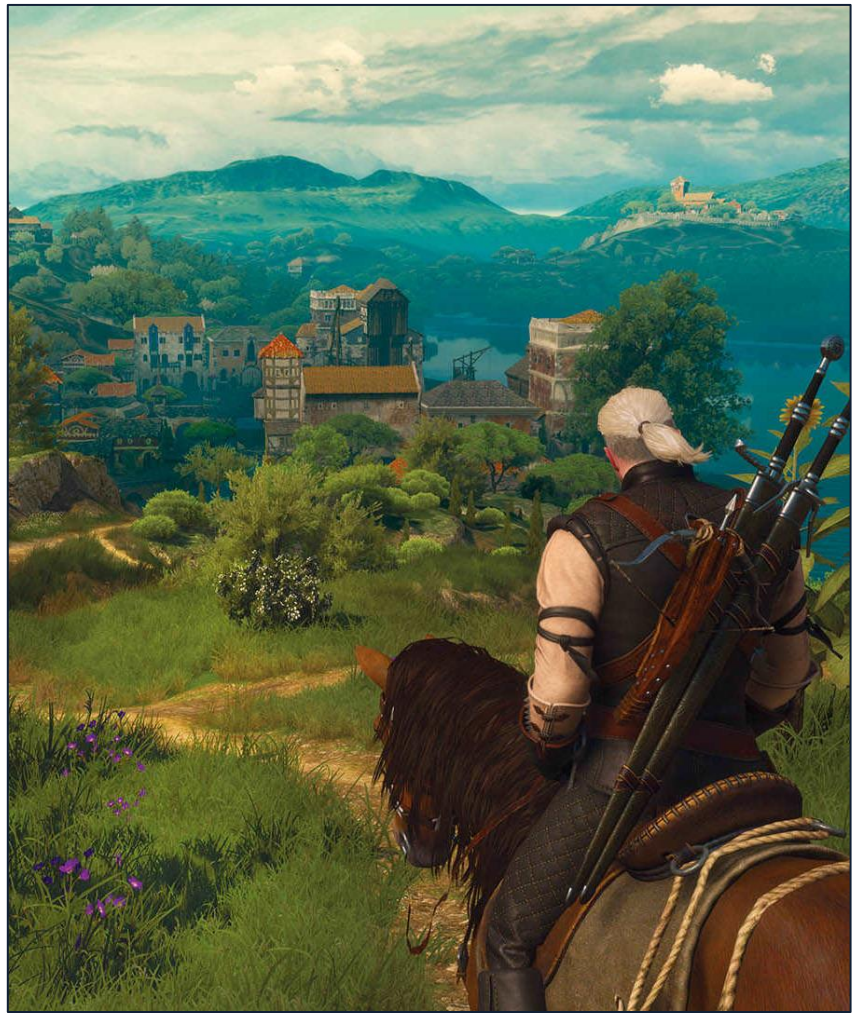
- An error is any action that does not accomplish the desired goal
- Error rate can be measured by counting the number of errors users commit while performing some task (as part of an experiment to measure also other usability attributes)
- Simply counting errors might be misleading
  - Some errors are corrected immediately by the users, and have the only effect of slowing they down (somewhat reducing the transaction rate)
  - Other errors are more catastrophic in nature
    - User does not notice, leading to a faulty work product
    - It may be impossible to recover from the error

Errors

- Users should make as few errors as possible when using a software
- And at least, they should make very few, if any, catastrophic errors!

Subjective Satisfaction

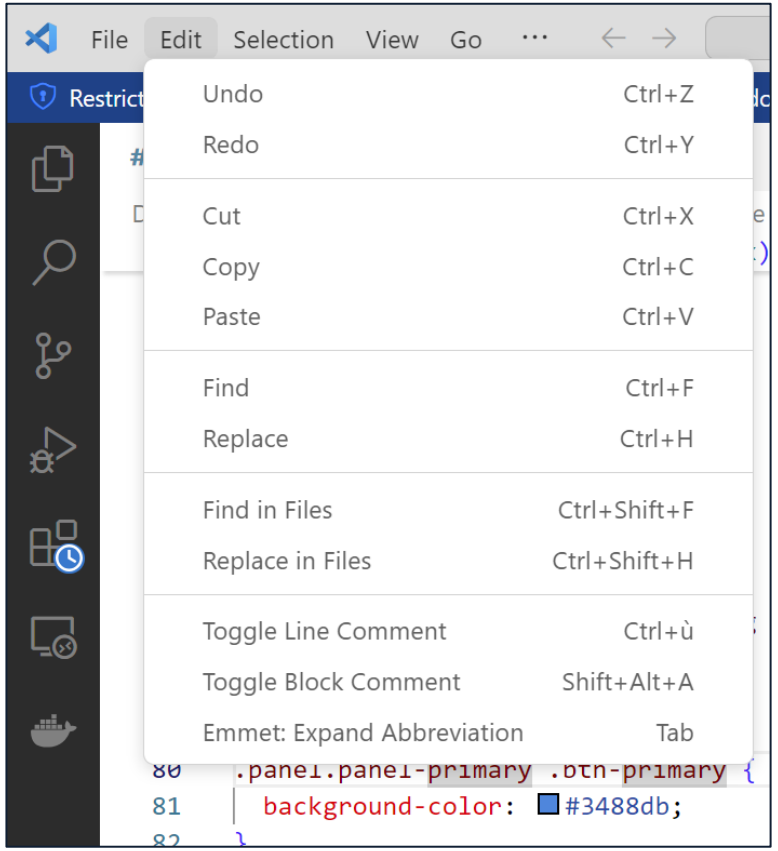
- This attribute refers to how **pleasant** it is to use the system
- It’s especially important for systems used on a discretionary basis
  - Videogames, creative painting, …
- For some such systems, their entertainment value is more important then the speed with which things get done, since one might want to spend a long(er) time having fun



Usability Trade-offs

It is not always possible to maximize all usability attributes simultaneusly

- **Trade-offs** could be necessary:
  - To avoid catastrophic errors, we may design a user interface that is less efficient, as it asks extra questions to ensure that the user really wants to perform a certain action
- In some cases, we may get a win-win:
  - Learnability and efficiency of use for experts are not necessarily conflicting
  - We may be able to get the best of both learning curves, for example by including ***accelerators*** (e.g.: shortcut or hotkeys) in our UI



Accelerators in VS Code

Usability Engineering

- Many software development projects fail to achieve their goals
  - Some estimate the failure rate as high as 50%!
- Much of these failures are due to poor communications between developers and clients and developers and users
- This results in UIs that force users to adapt and change their behaviour rather than accomodating the needs of the users

- So, how do we end up with producing usable (and useful) software?

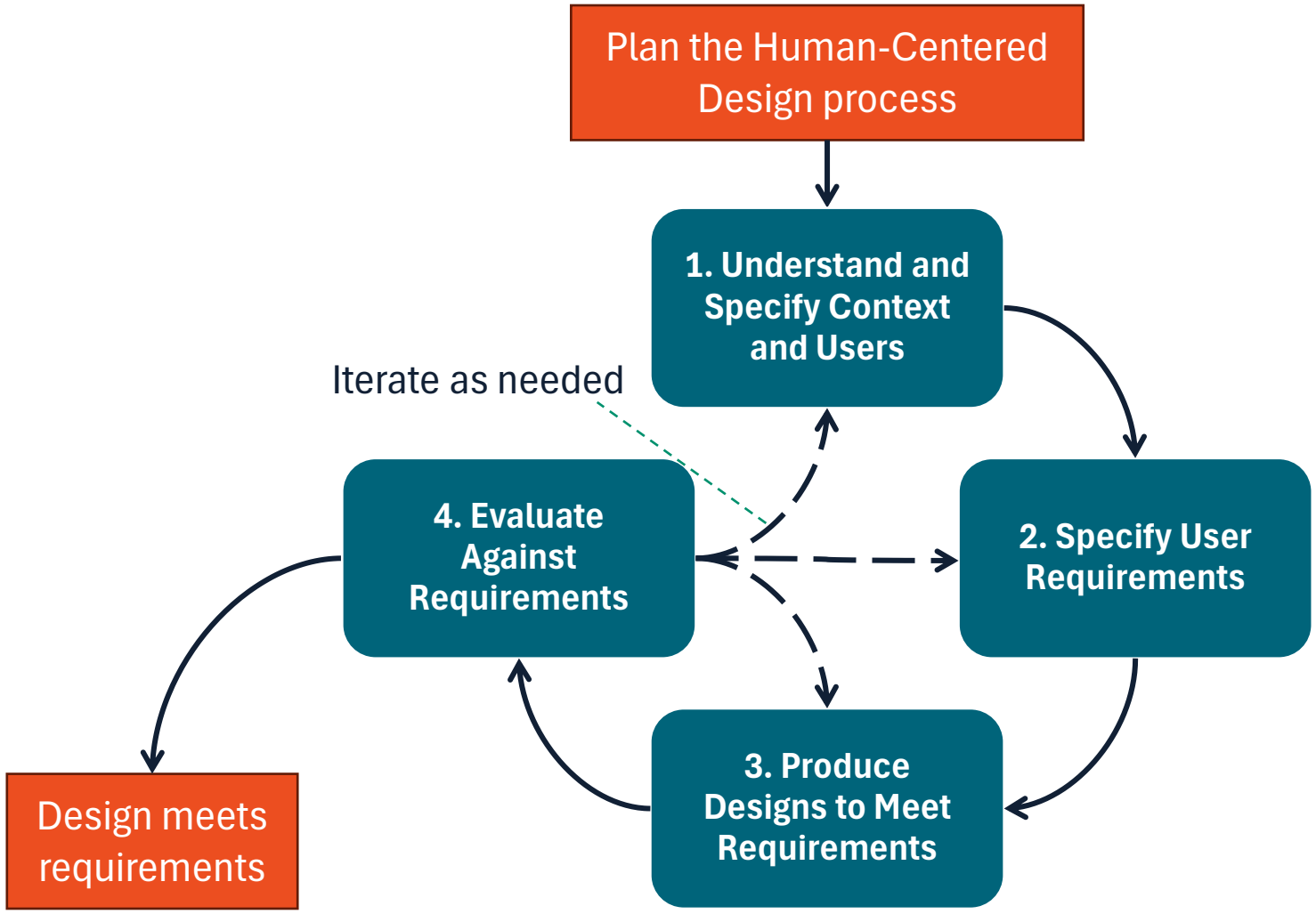
Human-Centered Design (HCD)

- Not a one-shot affair where the user interface is fixed up before release
- A set of activities that ideally take place throughout the Sofware Life Cycle
- ISO 9241-210 defines **Human-Centered Design (HCD)**
  - Developers need to keep a human-centered perspective
  - The users need to play a central role throughout the lifecycle
  - Complementary to existing design methodologies
  - Provides a human-centred perspective that can be integrated into different design and development processes

HCD Principles

- Design is based upon **explicit** understanding of users, tasks and evironments
- Users are **involved** as much as possible in design and development
- Design is driven and refined by **user-centered evaluation**
- The process can be **iterative**, if needed
- Design addresses the whole user experience
- The design team should include multidisciplinary skills and perspectives

HCD Activities



HCD: Understand Context and Users

Context: what are the type of uses of the system?

- Life-critical system?
- Industrial? Commercial? Military? Scientific?
- Entertainment?

What's the market the system competes in?

- Custom software development project?
- System for businesses?
- App for the general public?

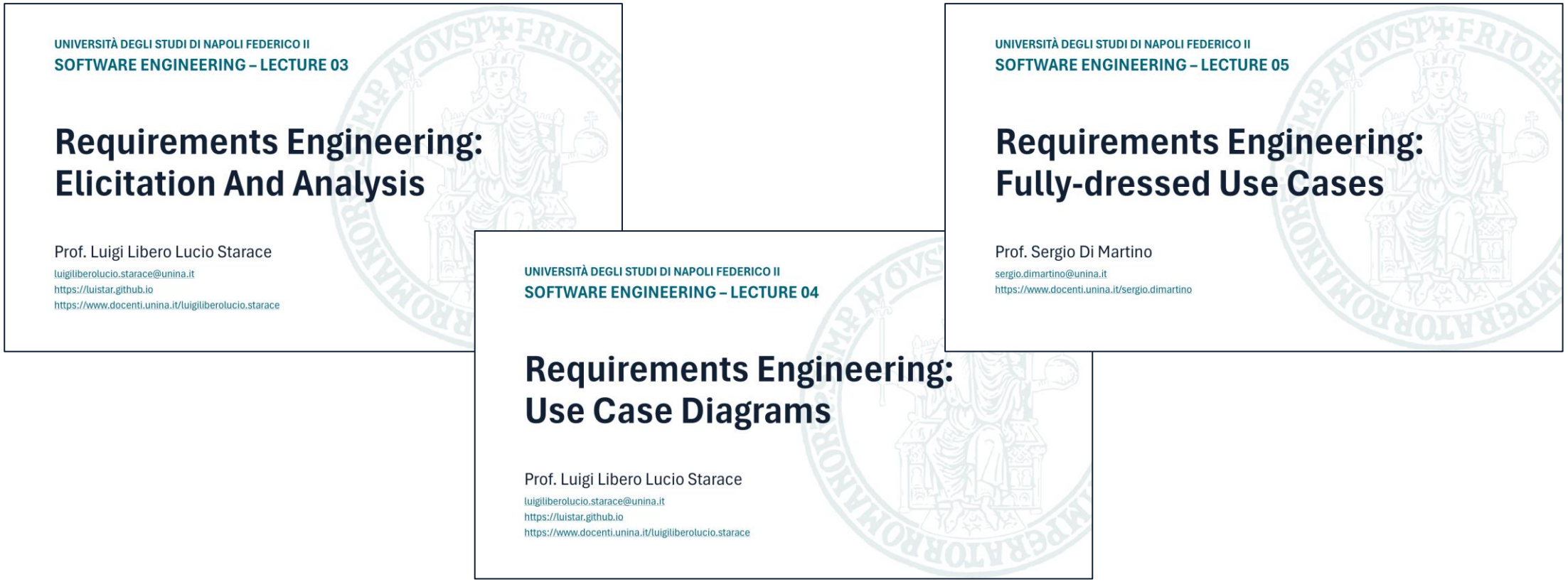
HCD: Understand Context and Users

Users: know thy users (as we did in Requirement Engineering)!

- **Personas** and **scenarios** (keep the needs of users always in mind)
- But also need to consider:
  - Physical attributes (age, gender, size, reach, visual angles, etc…)
  - Perceptual abilities (hearing, vision, heat sensitivity…)
  - Cognitive abilities (memory span, reading level, musical training, math…)
  - Physical work places (table height, sound levels, lighting, software version…)
  - Personality and social traits (likes, dislikes, preferences, patience…)
  - Cultural and international diversity (languages, dialog box flow, symbols…)
  - Special populations, (dis)abilities

HCD: Specify User Requirements

- Any idea how to do that?



HCD: Design to Meet Requirements

Appropriate design of the system relies on a clear understanding of context and users!

Producing design solutions should include the following sub-activities:

a) designing user tasks, user-system interaction and user interface to meet user requirements, taking into consideration the whole user experience;
b) making the design solutions more concrete (e.g.: prototypes or mock-ups);
c) improving design solutions based on user-centred evaluation and feedback
d) communicating the design solutions to those responsible for their implementation.

HCD: Design to Meet Requirements

- Designing the **user-system interaction** involves deciding how users will accomplish tasks **with** the system rather than describing what the system looks like.
- For example, decisions at this point can include issues such as the
  - choice of **modality** (e.g. auditory, visual and tactile)
  - choice of **media** (e.g. text versus graphics, dialogue boxes versus wizards, mechanical versus electronic controls)

HCD: Design to Meet Requirements

- Designing the interaction should include:
  - making high-level decisions (e.g. initial design concept, essential outcomes);
  - identifying tasks and sub-tasks;
  - allocating tasks and sub-tasks to the user and to other parts of the system;
    - E.g.: system keeps track of login id and reminds users, but users remember the password
  - identifying the interaction objects required for the completion of the tasks;
  - designing the sequence and timing (dynamics) of the interaction;
  - designing the UI to allow efficient access to interaction objects.

HCI: Evaluating the Design

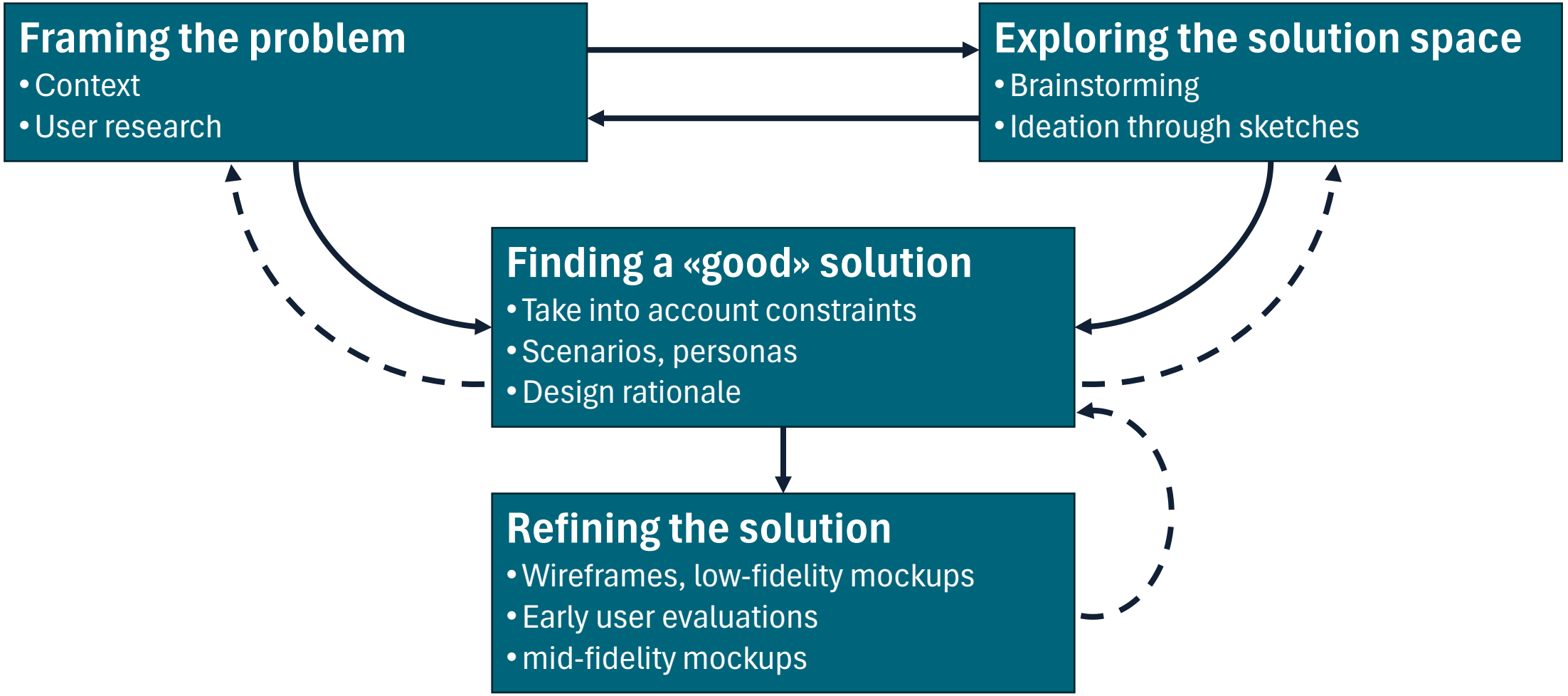
User-centered evaluation (i.e., evaluation based on user perspective) is a required activity in HCD:

- **User-based testing** (e.g.: involving real or representative users)
- **Inspection-based approach** (checking guidelines or requirements)

Human-Centered Design and the SDLC

- HCD does not require any particular design process
- It is complementary to existing development methodologies
- Each activity can be integrated (to a lesser or greater extent) at any stage in the development of a system
- For example, HCD could be applied in the Requirement Engineering stage in a waterfall process model

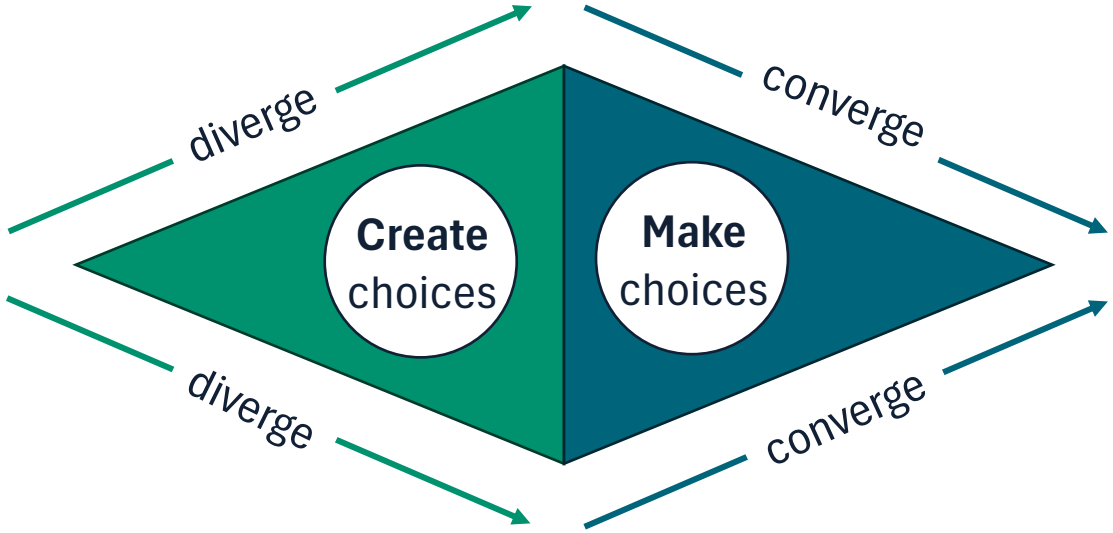
More on the UI Design Process



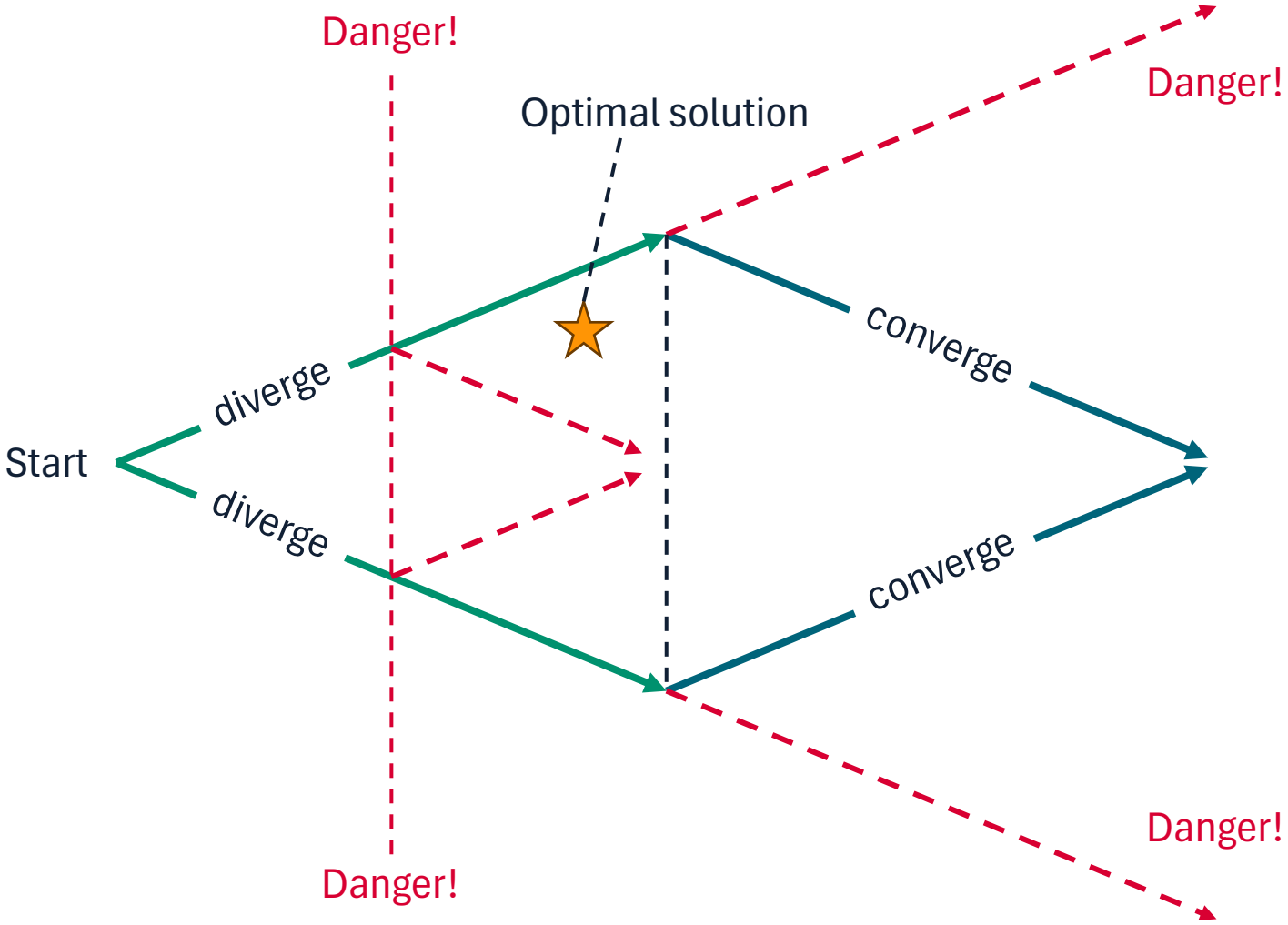
Design as Choices

The design process is an alternance of two phases:

- **Divergence** phase (elaboration)
  - Explore different designs
  - Create choices
- **Convergence** phase (reduction)
  - Choose among different designs



The Design Diamond



- Stopping divergence too early might lead to missing good ideas
- Keep in mind that we are operating within budget and organizational constraints: we need to actually converge!

Importance of Critique and Feedback

Ideas can be **good** or **bad**

- Both kind of ideas are **useful** in design
- By making clear what is a bad design, we can avoid implementing it
- Bad ideas can also help us justify our good ideas
- Feedback can further improve a good idea!

Tips for Giving Effective Critiques

Hamburger Method

- **Bun**
  - Fluffy and nice
- **Meat**
  - How to improve
- **Bun**
  - Fluffy and nice

I like, I wish, What if?

- **I like…**
  - Lead with something nice
- **I wish…**
  - Something you would improve
- **What if…?**
  - An idea to spark further discussion

Socratic Method

- **Identify one aspect of design and ask «why?»**
  - Forces presenter to give, or develop, motivations for design decisions
  - Not inherently negative, hard to get defensive

Tips for Giving Effective Critiques

- Limit the use of personal pronouns (e.g.: «you»)
  - Critique is about the artifact, not the designer
- A designer deserves honest feedback
  - Be honest, give both positive and negative feedback
  - Be clear and motivate your critiques
- Help with actionable suggestions