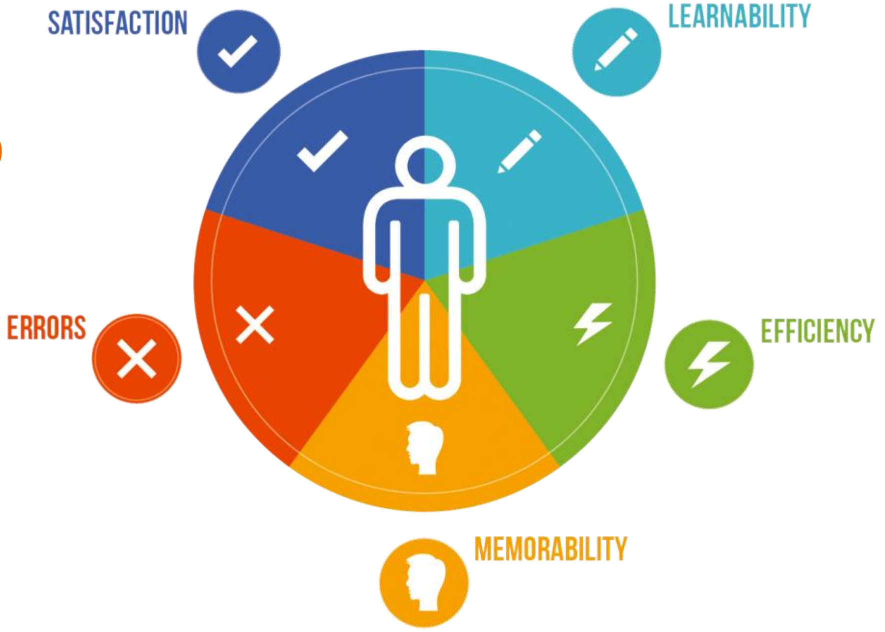
User-centered design

GUIDELINES FOR DESIGN AND EVALUATION

REQUIREMENTS IDENTIFICATION



POLITECNICO DI TORINO

Laura Farinetti - DAUIN

e-Lite

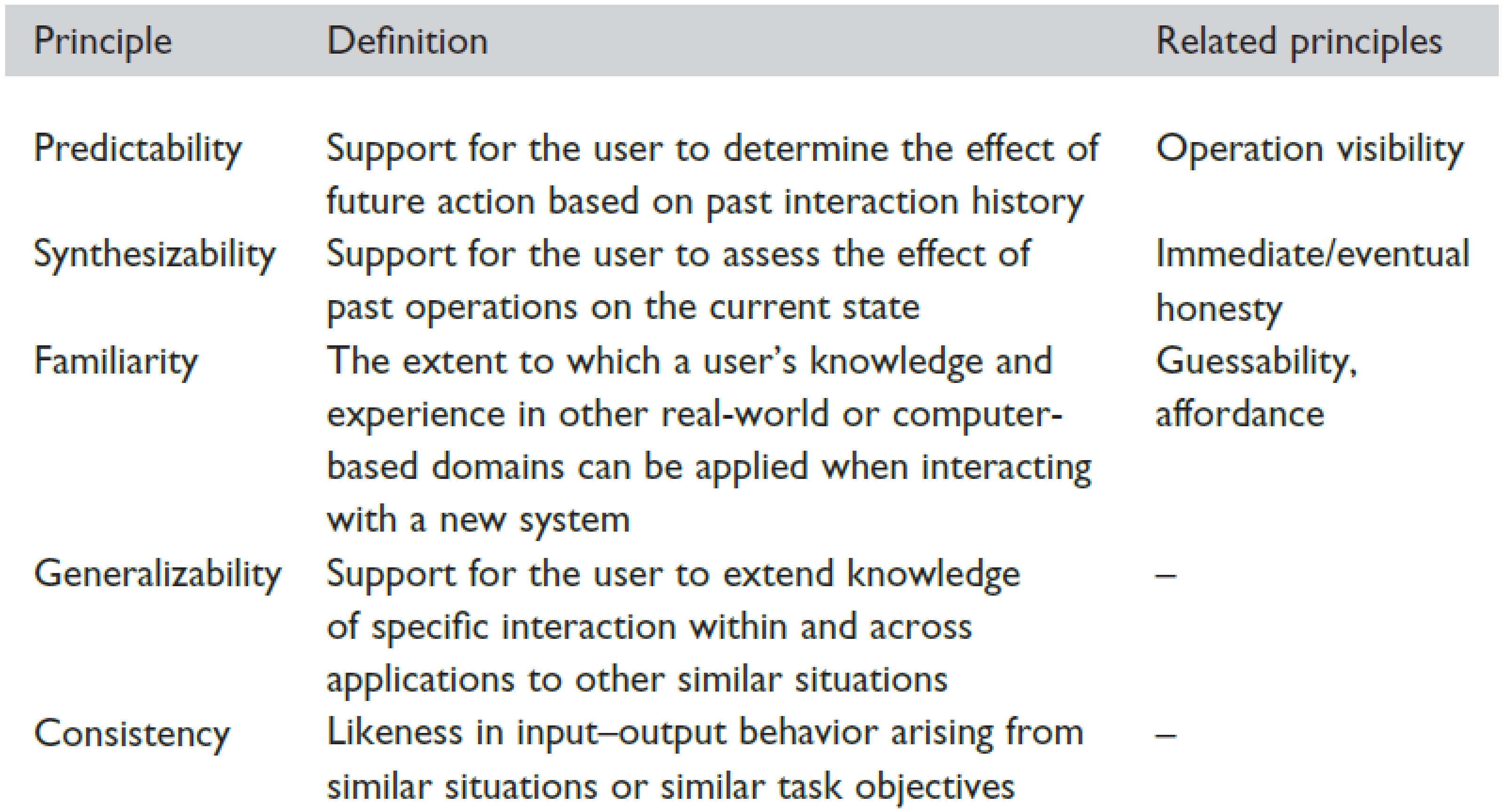
Design for usability

- One of the central problems that must be solved in a user-centered design process is how to provide designers with the ability to determine the usability consequences of their design decisions
- Design rules, classified along two dimensions: authority and generality
  - Principles: abstract design rules, with high generality and low authority
  - Standards: specific design rules, high in authority and limited in application
  - Guidelines: lower in authority and more general in application
- Can be supported by psychological, cognitive, ergonomic, sociological, economic or computational theory, which may or may not have roots in empirical evidence

Principles to support usability

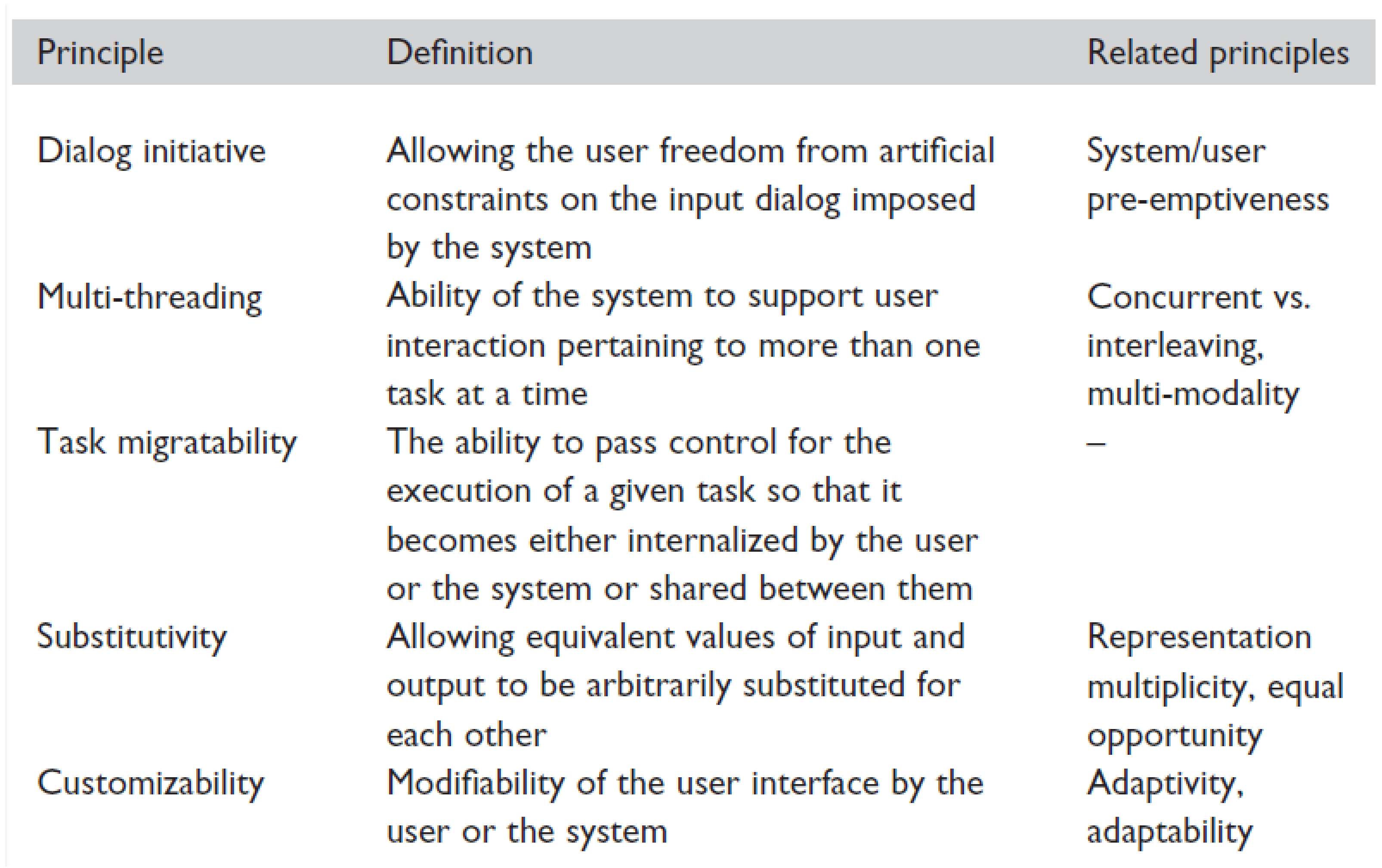
- Three main categories
  - Learnability: the ease with which new users can begin effective interaction and achieve maximal performance
  - Flexibility: the multiplicity of ways in which the user and system exchange information
  - Robustness: the level of support provided to the user in determining successful achievement and assessment of goals

Learnability



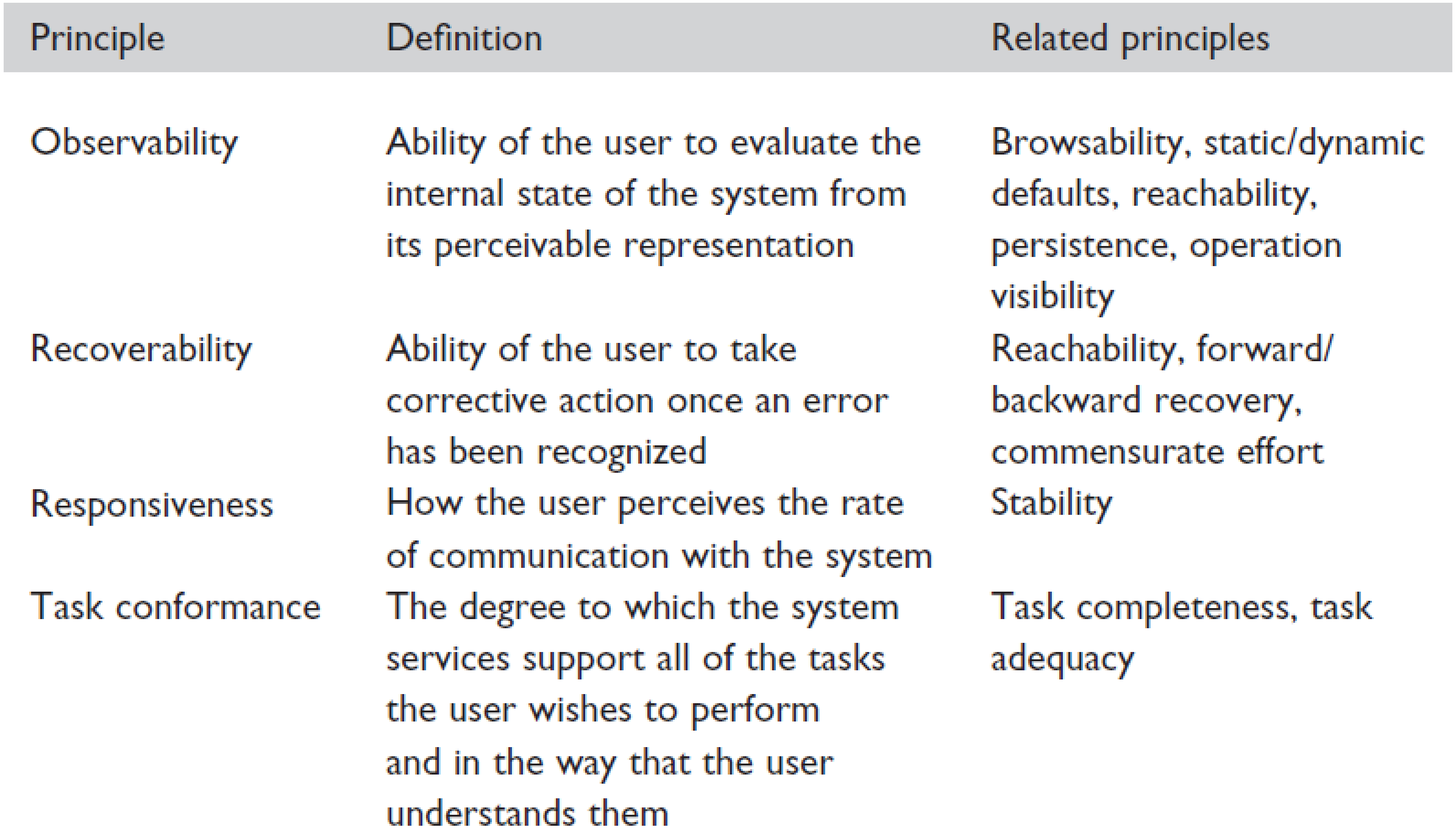
Principle	Definition	Related principles
Predictability	Support for the user to determine the effect of future action based on past interaction history	Operation visibility
Synthesizability	Support for the user to assess the effect of past operations on the current state	Immediate/eventual honesty
Familiarity	The extent to which a user's knowledge and experience in other real-world or computer-based domains can be applied when interacting with a new system	Guessability, affordance
Generalizability	Support for the user to extend knowledge of specific interaction within and across applications to other similar situations	–
Consistency	Likeness in input–output behavior arising from similar situations or similar task objectives	–

Flexibility



Principle	Definition	Related principles
Dialog initiative	Allowing the user freedom from artificial constraints on the input dialog imposed by the system	System/user pre-emptiveness
Multi-threading	Ability of the system to support user interaction pertaining to more than one task at a time	Concurrent vs. interleaving, multi-modality
Task migratability	The ability to pass control for the execution of a given task so that it becomes either internalized by the user or the system or shared between them	–
Substitutivity	Allowing equivalent values of input and output to be arbitrarily substituted for each other	Representation multiplicity, equal opportunity
Customizability	Modifiability of the user interface by the user or the system	Adaptivity, adaptability

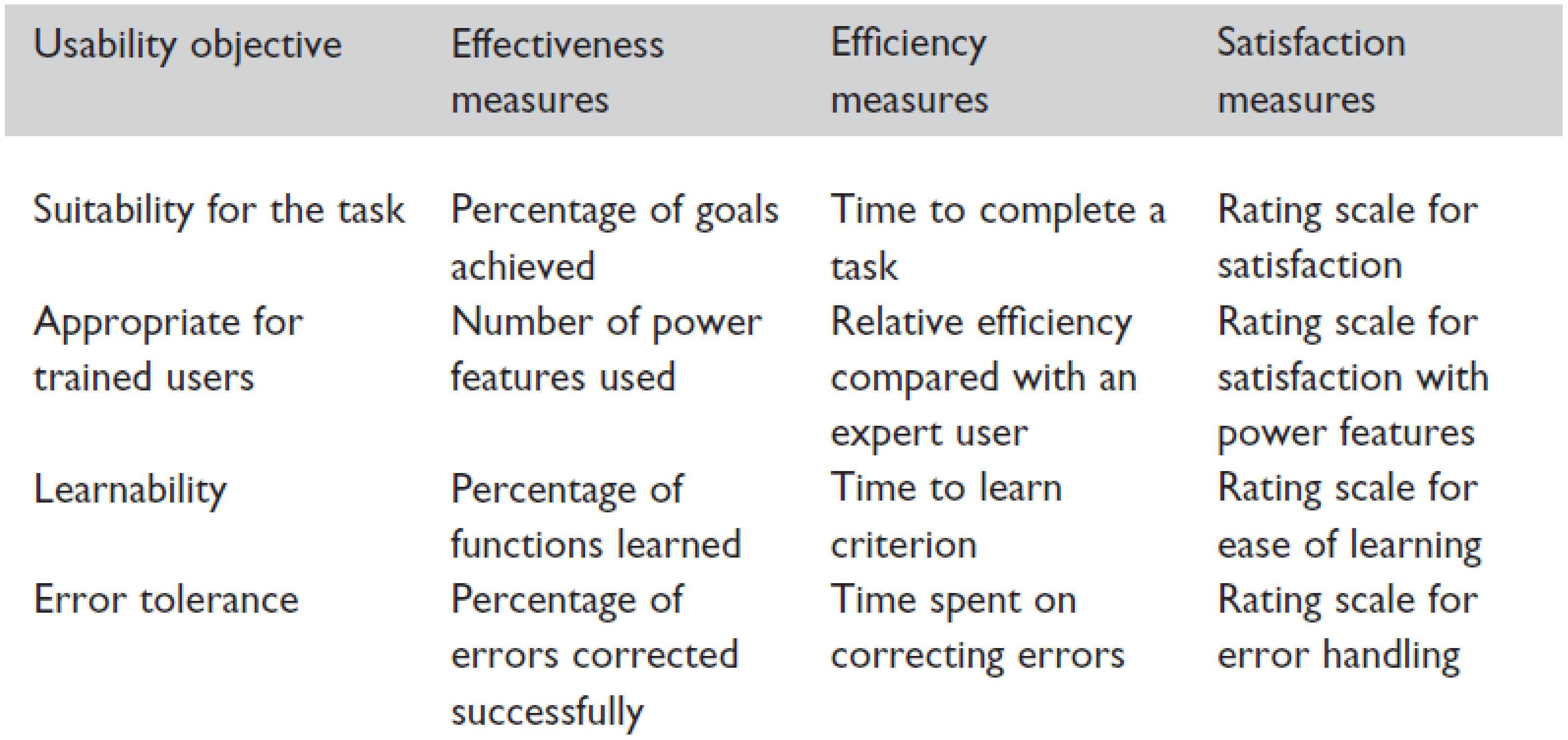
Robustness



Principle	Definition	Related principles
Observability	Ability of the user to evaluate the internal state of the system from its perceivable representation	Browsability, static/dynamic defaults, reachability, persistence, operation visibility
Recoverability	Ability of the user to take corrective action once an error has been recognized	Reachability, forward/backward recovery, commensurate effort
Responsiveness	How the user perceives the rate of communication with the system	Stability
Task conformance	The degree to which the system services support all of the tasks the user wishes to perform and in the way that the user understands them	Task completeness, task adequacy

Usability metrics from ISO 9241

- Ergonomic Requirements for Office Work with Visual Display Terminals (VDT)s



Usability objective	Effectiveness measures	Efficiency measures	Satisfaction measures
Suitability for the task	Percentage of goals achieved	Time to complete a task	Rating scale for satisfaction
Appropriate for trained users	Number of power features used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Learnability	Percentage of functions learned	Time to learn criterion	Rating scale for ease of learning
Error tolerance	Percentage of errors corrected successfully	Time spent on correcting errors	Rating scale for error handling

Golden rules and heuristics

- Nielsen’s ten heuristics (mainly for evaluation)
- Shneiderman’s eight golden rules
- Norman’s seven principles

Shneiderman’s eight golden rules of interface design

- Intended to be used during design, but can also be applied to evaluation

1. Strive for consistency in action sequences, layout, terminology, command use and so on
2. Enable frequent users to use shortcuts, such as abbreviations, special key sequences and macros, to perform regular, familiar actions more quickly.
3. Offer informative feedback for every user action, at a level appropriate to the magnitude of the action
4. Design dialogs to yield closure so that the user knows when they have completed a task

Shneiderman’s eight golden rules of interface design

5. Offer error prevention and simple error handling so that, ideally, users are prevented from making mistakes and, if they do, they are offered clear and informative instructions to enable them to recover
6. Permit easy reversal of actions in order to relieve anxiety and encourage exploration, since the user knows that he can always return to the previous state
7. Support internal locus of control so that the user is in control of the system, which responds to his actions
8. Reduce short-term memory load by keeping displays simple, consolidating multiple page displays and providing time for learning action sequences

Norman’s seven principles for transforming difficult tasks into simple ones

1. Use both knowledge in the world and knowledge in the head
   - People work better when the knowledge they need to do a task is available externally, but experts also need to be able to internalize regular tasks to increase their efficiency
   - Systems should provide the necessary knowledge within the environment and their operation should be transparent to support the user in building an appropriate mental model of what is going on
2. Simplify the structure of task
   - Tasks need to be simple in order to avoid complex problem solving and excessive memory load
3. Make things visible: bridge the gulfs of execution and evaluation
   - The interface should make clear what the system can do and how this is achieved, and should enable the user to see clearly the effect of their actions on the system

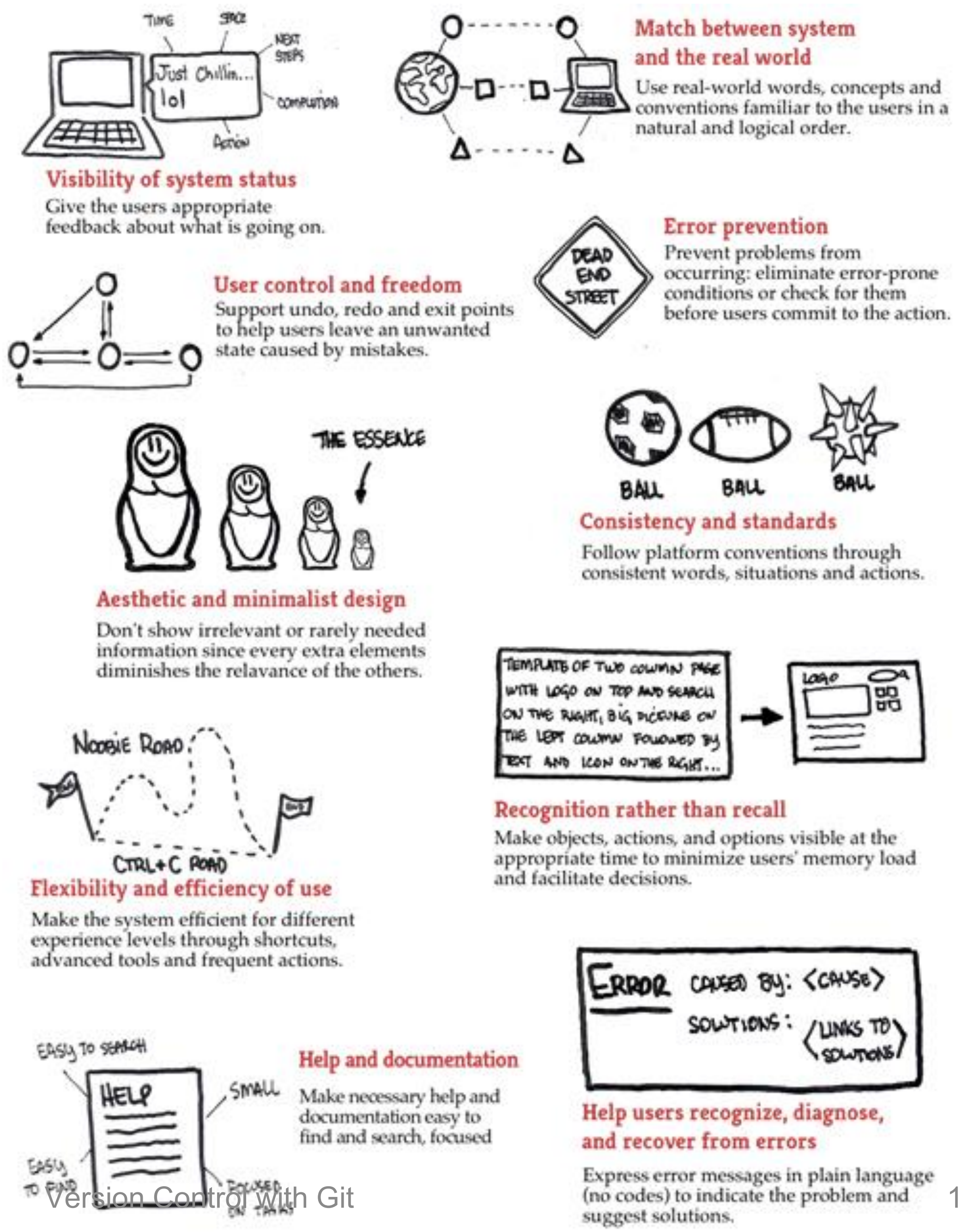
Norman’s seven principles for transforming difficult tasks into simple ones

4. Get the mappings right
   - User intentions should map clearly onto system controls
   - User actions should map clearly onto system events
   - Controls, sliders and dials should reflect the task (e.g. a small movement has a small effect and a large movement a large effect)
5. Exploit the power of constraints, both natural and artificial
   - Constraints are things in the world that make it impossible to do anything but the correct action in the correct way (e.g puzzles)

- Design for error
  - Anticipate the errors the user could make and design recovery into the system
- When all else fails, standardize
  - If there are no natural mappings then arbitrary mappings should be standardized so that users only have to learn them once
  - E.g., in cars the critical controls (accelerator, brake, clutch, steering) are always the same

Nielsen's ten heuristics

- Broad rules (1995)



Visibility of system status
Give the users appropriate feedback about what is going on.

Match between system and the real world
Use real-world words, concepts and conventions familiar to the users in a natural and logical order.

User control and freedom
Support undo, redo and exit points to help users leave an unwanted state caused by mistakes.

Error prevention
Prevent problems from occurring: eliminate error-prone conditions or check for them before users commit to the action.

Consistency and standards
Follow platform conventions through consistent words, situations and actions.

Aesthetic and minimalist design
Don't show irrelevant or rarely needed information since every extra elements diminishes the relavance of the others.

Recognition rather than recall
Make objects, actions, and options visible at the appropriate time to minimize users' memory load and facilitate decisions.

Flexibility and efficiency of use
Make the system efficient for different experience levels through shortcuts, advanced tools and frequent actions.

Help and documentation
Make necessary help and documentation easy to find and search, focused

Help users recognize, diagnose, and recover from errors
Express error messages in plain language (no codes) to indicate the problem and suggest solutions.

Formalizing requirements

- The initial vision and user inputs must be “distilled” into a set of requirements
- Strategic choices: what is in, what is out
- Describes what the system does, and the external constraints
- Might be used as a “specification contract”
- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- Requirements engineering
  - The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed

Types of requirements

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints
  - Written for customers
- System requirements (or developer requirements)
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints
  - Defines what should be implemented so it may be part of a contract between client and contractor

Example

User requirement definition

The software must provide a means of representing and accessing external files edited by other tools

System requirements specification

1.1 The user should be provided with facilities to define the type of external files

1.2 Each external file type may have an associated tool which may be applied to the file

1.3 Each external file type may be represented as a specific icon on the user’s display

1.4 Facilities should be provided for the icon representing an external file type to be defined by the user

1.5 When a user selects an icon representing an external file the effect of that selection is to apply the tool associated with the external file type to the file represented by the selected icon

Types of requirements

- Functional requirements (FR)
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations
- Non-functional requirements (NFR)
  - Aka Quality requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain

Functional Requirements (FR)

- What the system does
- What functions it offers to its users
- Don’t care how they will be implemented (yet)
- A long list of “local” features (easy to identify a small portion of the system that delivers that function)

Examples

- FR3.1: The user must be able to activate and de-activate the wake-up service. This decision will be applied until the user changes it again.
- FR3.2: The user must be able to silence the wake-up service just for the next day. Service will resume automatically on the following day.
- FR4.4: The user must be able to set up an “ad hoc” wake-up call, that will run only once, will not be remembered, and will have specific settings
- FR4.4.1: The user may configure the settings of any already defined “ad hoc” call
- FR4.4.2: The user may configure the default settings for (to be created) “ad hoc” calls

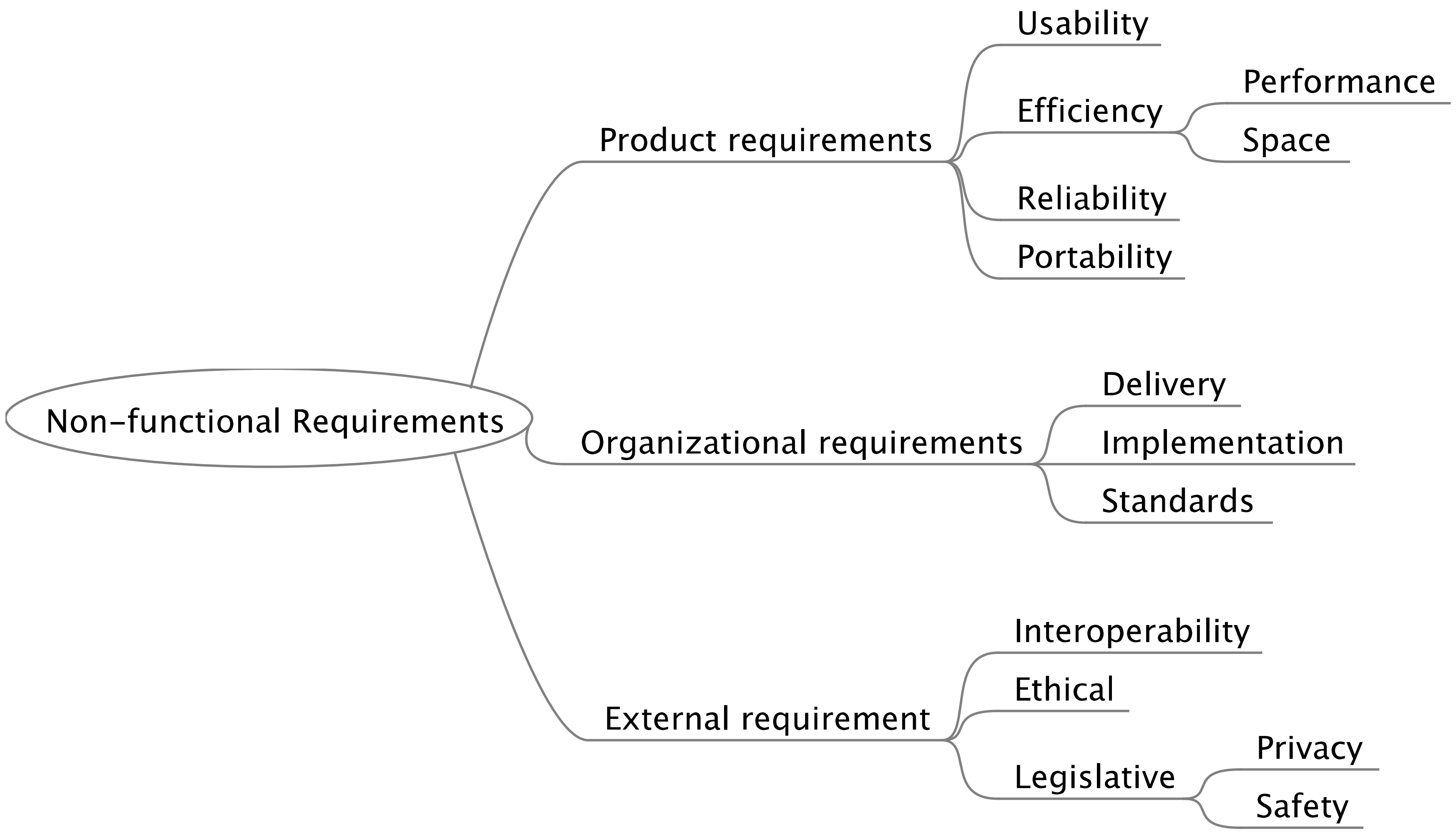
Non-functional requirements (NFR)

- Define system properties and constraints e.g. reliability, response time and storage requirements.
  - Constraints are I/O device capability, system representations, Supported devices, Usability, Language, etc.
- Process requirements may also be specified mandating a particular set of tools, programming language or development method
- Non-functional requirements may be more critical than functional requirements: if these are not met, the system is useless.

Pervasiveness in NFR

- NFR are usually “general” and cannot be localized to a single spot in system implementation
- Every function, in every module, in every screen, in every device, …. must guarantee that no NFR is broken

Non-functional requirements



Examples

- NFR1: The mobile interfaces must be compatible with iOS (8.0 and later), Android (4.2 and later)
- NFR2: The system will be localized in many languages (default: English)
- NFR18: The system should work, in reduced conditions, even if the user mobile device is switched off or disconnected
- NFR3: The web interfaces will be compatible with browsers … version ….
- NFR4: The web interfaces will be “responsive”, and will adapt to screen resolutions from 800x600 to 1920x1080

Good requirements

- The best requirements are

Complete
Correct
Clear
Verifiable
Necessary
Feasible
Prioritized
Consistent
Traceable
Modular
Design-independent

Good requirements

- Complete: express a whole idea or statement
  - Should describe completely the user task and the information required to support the task
  - Define response to all possible inputs (both correct and incorrect)
  - Define all terms and unit of measure
  - Focusing on system functionality instead of user needs to be accomplished may lead to incomplete requirements
- Example

‘We must be able to change an employee’s profile information’

‘We must be able to change the employees last name, first name, middle initial, street address, city, state, zip code, marital status’

Good requirements

- Correct: technically and legally possible
  - The requirements should be appropriate to meet the goals of the project and accurately describe the user's expectations of the functionality
  - Customer or users can determine if the requirement correctly reflects their actual needs
- Example

'Employees only change their name when their address or their marital status changes'

'Employees may change their name in the payroll system by providing the appropriate legal proof of the change. The change may come with a change in marital status, address or be made alone'

Good requirements

- Clear: unambiguous and not confusing
  - Requirements should be written so that all readers will arrive at a single, consistent interpretation
  - Clear to those who create it and to those who use it
  - Ambiguous requirements can result in the wrong system being developed and may not be found during testing due to the incorrect interpretation of the requirements
- Example

‘Employees are not allowed to work for more than 80 hours in one week’

‘Employee time worked: the time worked is recorded in hours, the smallest increment recorded is .25 of an hour. If an employee reports more than 80 hours in a 7 day period, a warning is provided to the supervisor and the payment is held for approval.’

Good requirements

- Verifiable: it can be determined whether the system meets the requirement
  - Each requirement should be testable and verifiable
  - There exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement
  - Ambiguous requirements are not verifiable
- Example

‘The system should be easy to use’

‘A novice user must be able to add a new employee to the payroll system within 10 minutes’

Good requirements

- Necessary: should support one of the project goals
  - Related to specific and meaningful goals
- Example

‘We should be able to enter the employee eye colour’

↓

Why is this requirement necessary?

Good requirements

- Feasible: it can be accomplished within cost and schedule
  - The business analyst must be sure that all requirements are technologically possible for a reasonable cost
- Example

'The system should automatically be updated when the government changes the law'

⇩

Although this requirement may be technologically feasible, it would involve a complex interface (and likely new government system) with an outside organisation which would be very expensive and difficult to negotiate.
Is it a critical requirement?

Good requirements

- Prioritized: tracked according to business need levels
  - Each requirement should be prioritised
- Most organisations use the MoSCoW method for prioritisation
  - Must Have: the system must meet this requirement for the end product to be considered a success
  - Should Have: the system should have this requirement for it to solve the main business problem
  - Could Have: it would be good to include this requirement to ensure maximum benefit
  - Would Have: this is a nice to have requirement which the business could do without if necessary

Good requirements

- Consistent: not in conflict with other requirements
  - No subset of requirements is in conflict
  - Logical and temporal consistence
- Traceable: uniquely identified and tracked
  - Backward: explicitly referencing source in earlier documents
  - Forward: unique name or reference number

Good requirements

- Modular: can be changed without excessive impact
  - Structure and style such that any changes can be made easily, completely, and consistently while retaining the structure and style
  - Well structured, non redundant, separate requirements
- Design-independent: do not pose specific solutions on design

License