POISE: A Framework for Designing Perfect Interactive Systems with and for Imperfect People

Philippe Palanque

POISE: A Framework for Designing Perfect Interactive Systems with and for Imperfect People

Philippe Palanque [0000-0002-5381-971X]

ICS-IRIT, Université Toulouse III-Paul Sabatier, Toulouse, France
palanque@irit.fr

Abstract. The operator is frequently considered as the main sources of vulnerability in command and control systems; for example, in a 2006 survey 79% of fatal accidents in aviation were attributed to "human error." Beyond the case of command and control systems, users' faults occur not only at use time but also during the design and development of systems. Following Avizienis et al.'s taxonomy for faults, human-made error can be characterized as the operator's failure to deliver services while interacting with the interactive system. Non human-made errors are called natural faults and may occur during development or set the interactive system as well as its users into an error-state during its use. Focusing on interactive systems specificities, this paper presents a comprehensive description of faults covering both development and operation phases. In correspondence with this taxonomy, we present mechanisms to avoid, remove, tolerate and mitigate faults in order to design and develop what we call "perfect" interactive systems taking into account the organization, the interactive system, the environment and the people operating them. We define an interactive system as perfect when it blends multiple and diverse properties such as usability, security, user experience, dependability, learnability, resilience … We present multiple concrete examples, from aviation and other domains, of faults affecting socio-technical systems and associated fault-tolerant mechanisms.

Keywords: Faults, Interactive Systems, Dependability, Usability, Security, User Experience.

1 Introduction

Over the last decades, the research work in the field of HCI has been focussing on supporting user-related properties such as **usability** [90], **privacy** [77], **accessibility** [9] or **user experience** [54]. Contributions have been ranging from increased understanding of human behaviour (e.g. motor side [1], perceptive side [63] or cognition [4]) to the design of interaction techniques and innovative input and output devices. Unfortunately, as pointed out in [8] these contributions are rarely incorporated into products that are designed using early understanding of human behaviour (e.g. [23]) and standardized interaction (e.g. IBM CUA [13]) incorporated in Operating Systems manufacturers (e.g. touch interactions for Android[3]). While [8] argues that this can be solved by changing the focus of HCI research from user interfaces to interaction, this paper

argues that these research contributions should take more into account (an in an integrated way):

- the People (performing the tasks and the work),
- the Interactive System (that is used to perform the work),
- the Organization (providing the work context for the People and being the project owner of the Interactive System)
- the environment (where the People work and where the Interactive System is deployed.

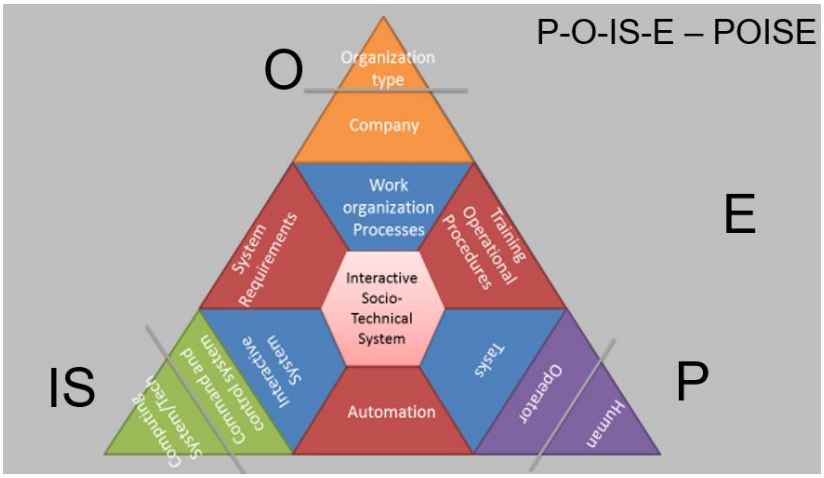

Fig. 1. The POISE framework blending People, Organization, Interactive System and the Environment

The three vertexes of the triangle represent the three main components POISE framework. These elements are connected to the other ones by dedicated trapeziums (Automation, System Requirements and Training and Operational Procedures). These trapeziums represent explicitly how the element influences the other ones. At the basis of the vertexes of the triangles, the blue trapeziums refine the description of the content of the vertexes. This way, Work organization and Processes are refinement of the organization, Tasks are refinement of the People and Interactive Systems are the relevant part of the Technology component. This is a refinement of the early work from Meshkati [64], claiming that the resilience of socio-technical systems require addressing in the same single framework Human Organization and Technology. Outside of the triangle the grey part corresponds to the Environment where the interactive system is deployed and where the people are working. The environment may be highly dynamic like weather condition for an aircraft or very static and controlled like a dark room of an enroute air traffic control centre.

While the four aspects of POISE have to be taken into account holistically, this paper leaves out aspects related to the organization including standards, training and requirements. Indeed, the focus is here about interaction technologies and their users but taken

explicitly into real life concerns imposed by the operational environment. In this context, other properties are relevant (and sometimes of higher importance) that the user-related ones mentioned above. Properties such as **reliability** [59], **dependability** [7], **resilience** [88], **fault-tolerance** [38] or **security** [56] are "usually" related to the interactive system element of POISE while they can also apply to the entire socio-technical systems (including the organization) as argued in [49].

We believe the approaches proposed in this paper would benefit any deployable interactive system (including desktop application or entertainment interactive software) but the benefits are more tangible in the context of safety critical ones.

A safety-critical system is a system in which failures or errors potentially leads to loss of life or injuries of human beings [19] while a system is considered as critical when the cost of a potential failure is much higher than the development cost. Whether or not they are classified as safety-critical or "only" critical, interactive systems have made their way into most of the command and control workstations including satellite ground segments, military and civil cockpits, air traffic control... Furthermore, the complexity and quantity of data manipulated, the amount of systems to be controlled and the high number of commands to be triggered in a short period of time have required the design, development and deployment of sophisticated interaction techniques.

Building reliable and dependable interactive systems is a cumbersome task due to their very specific nature. The behaviour of these reactive systems is event-driven. As these events are triggered by human operators manipulating hardware input devices, these systems have to react to unexpected events. On the output side, information (such as the current state of the system) has to be presented to the operator in such a way that it can be perceived and interpreted correctly. Lastly, interactive systems require addressing together hardware and software aspects (e.g., input and output devices together with their device drivers).

In the domain of fault-tolerant systems, empirical studies have demonstrated (e.g., [67]) that software crashes may occur even though the development of the system has been extremely rigorous. One of the many sources of such crashes is called natural faults [7] triggered by alpha-particles from radioactive contaminants in the chips or neutron from cosmic radiation. A higher probability of occurrence of faults [82] concerns systems deployed in the high atmosphere (e.g., aircrafts) or in space (e.g., manned spacecraft [48]). Such natural faults demonstrate the need to go beyond classical fault avoidance at development time (usually brought by formal description techniques and properties verification) and to identify all the threats that can impair interactive systems.

The paper is structured as follows. The next section focuses on the identification of the of the specificities of interactive systems. It presents the H-MIODMIT architecture which extends MIODMIT [29] architecture by incorporating the operator. The third section focuses introduces two classifications of faults one dedicated to faults altering the functioning of the interactive system and the other one dedicated to faults altering the behaviour of the operator. The fourth section identifies processes and tools that, when combined, can contribute to the quest for perfect interactive systems by providing means of incorporating and evaluating the presence (or absence) of the properties men-

tioned above. The fifth section illustrates contributions from the HCI, dependable computing and software engineering domains that support some of the properties and how, by integrating them, they improve the overall quality of interactive systems. Last section concludes the paper and identifies possible paths towards perfect interactive systems.

2 Specificities of Interactive Systems and Their Users

2.1 The H-MIODMIT Generic Architecture

Fig. 2 presents an architectural view (from left to right) of the operator, the interactive command and control system, and the underlying system (e.g., an aircraft engine). This architecture is a simplified version of MIODMIT (Multiple Input and Output Devices and Multiple Interaction Techniques), a generic architecture for multimodal interactive systems [29] described in AADL [40]. Following the attribute dimensions of [7] we highlight (top right of **Fig. 2**) the hardware and software components, and show how the human operator interacts with them (thick dotted lines).

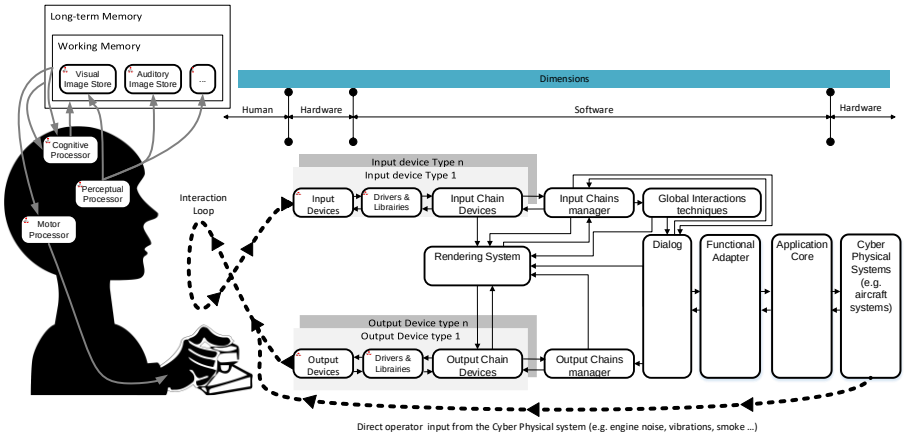

Fig. 2. H-MIODMIT architecture (adapted from [29])

As shown in the figure, interaction mainly takes place though the manipulation of input devices (e.g., keyboard or mouse) and the perception of information from the output devices (e.g., a computer screen or speaker). Another channel usually overlooked is the direct perception by the operator of information produced (usually as a side effect) of the underlying cyber-physical systems (e.g., noise or vibrations from an aircraft engine (represented by the lower dotted line in the figure)).

The specificities of the Interaction. The top left of the Software section of the diagram corresponds to the interaction technique that uses information from the input devices. Interaction techniques have a tremendous impact on operator performance. Standard interaction techniques encompass complex mechanisms (e.g. modification of the cursor's movement on the screen according to the acceleration of the physical

mouse on the desk). This design space is of prime importance and HCI research has explored multiple possibilities for improving performance, such as enlarging the target area for selection on touch screens [69] and providing on-screen widgets to facilitate selection [2].

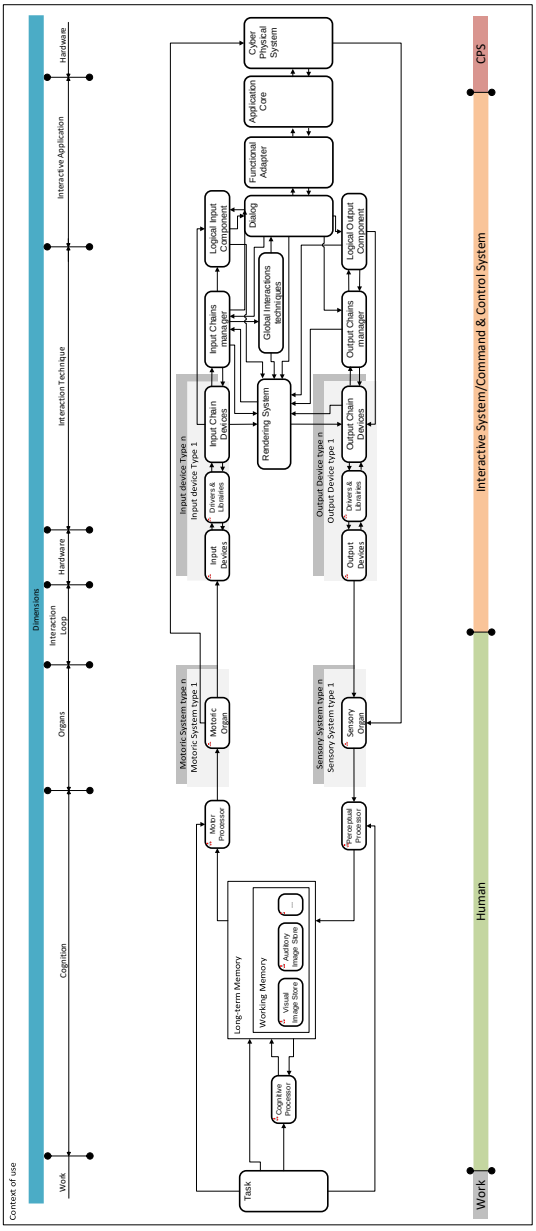

Fig. 3. H-MIODMIT detailed with the explicit representation of motor, perceptive and cognitive capability of operators and their tasks

The System side. The right side of the Software section of the architecture corresponds to what is usually called interactive applications. This is where HCI methods such as task analysis are needed for building usable application that fit the operators' work [32].

The Human side. The left side of **Fig. 2** represents the operator's view. The drawing is based on work that models the human as an *information processor* [24], based on previous research in psychology. In that model, the human is presented as a system composed of three interconnected processors. The *perceptive system* senses information from the environment – primarily the visual, auditory, and tactile systems as these are more common when interacting with computers. The *motor system* allows operators to act on the real world. Target selection (a key interaction mechanism) has been deeply studied [85]; for example, Fitts' Law provides a formula for predicting the time needed for an operator to select a target, based on its size and distance [43]. The *cognitive system* is in charge of processing information gathered by the perceptual system, storing that information in memory, analyzing the information and deciding on actions using the motor system. The sequential use of these systems (perceptive, cognitive and motoric) while interacting with computers is called the Human-Computer Interaction Loop (HCIL).

2.2 Incorporating Operators' Work

Fig. 3 proposes a refinement of H-MIODMIT presented in **Fig. 2**. The bottom of the figure adds description about the work of the operators (in term of tasks) and how this work is performed exploiting the motor, perceptive and cognitive processes described in [24]. This architecture fits POISE framework (see **Fig. 1**) as it covers entirely the bottom part of the triangle.

Describing users' tasks may be a complex and cumbersome activity especially when dealing with real domains [52]. Beyond, as shown in **Fig. 1**, this is where automation design takes place by migrating user's tasks to the interactive system. In addition, this design requires identifying the all the RCRAFT aspects: Responsibility, Resources, Authority and Control Transitions as defined in [44] and refined and connected to task models in [18].

3 Taxonomies of Faults

This section identifies the faults that can alter the functioning of the elements presented in the architecture presented in **Fig. 3**. We start by presenting the taxonomy of faults that impair the functioning of the interactive system and then present a recent taxonomy of faults that organizes the various types of faults that impair people's behaviour.

3.1 Faults Altering the System

To be able to ensure that the system will behave properly whatever happens, a system designer has to consider all the issues that can impair the functioning of that system. To this end the domain of dependable computing e.g. Avizienis et al [7] have defined a taxonomy of faults. This taxonomy leads to the identification of 31 elementary classes of faults. **Fig. 4** presents a simplified view of this taxonomy and makes explicit the two main categories of faults (top level of the figure): i) the ones made at development time (see left-hand side of the figure) including bad designs, programming errors, … and ii) the one made at operation times (see right-hand side of the figure) including operator errors such as slips, lapses and mistakes as defined in [80].

The leaves of the taxonomy are grouped into five different categories as each of them bring a special problem (issue) to be addressed:

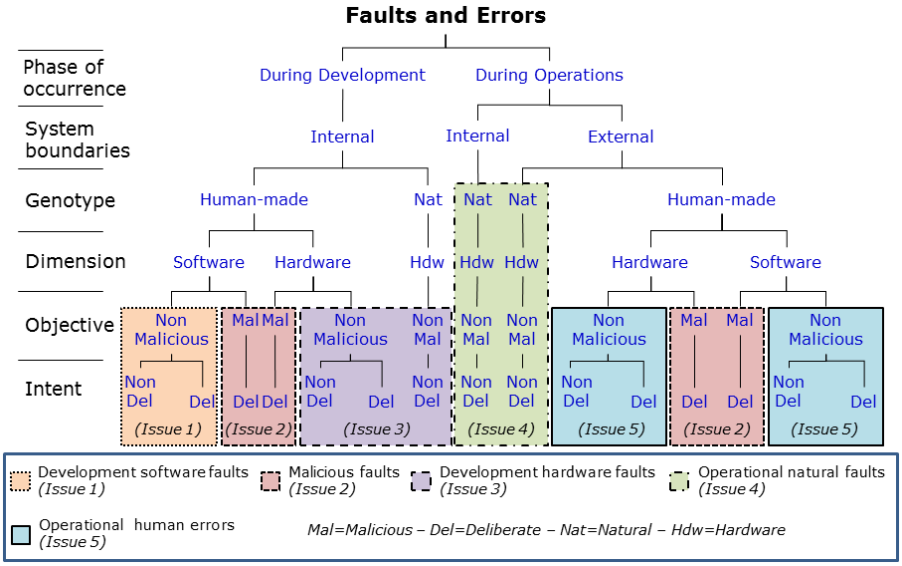

Fig. 4. Taxonomy of faults in computing systems (adapted from [7]) and associated issues for the dependability of these systems

- *Development software faults (issue 1):* software faults introduced by a human during system development. They can be, for instance, bad design errors, bugs due to faulty coding, development mistakes …
- *Malicious faults (issue 2):* faults introduced by human with the deliberate objective of damaging the system. They can be, for instance, an external hack causing service denial or crash of the system.
- *Development hardware faults (issue 3):* natural (e.g. caused by a natural phenomenon without human involvement) as well as human-made faults affecting the hardware during its development. They can be, for instance, a short circuit within a processor (due to bad construction).

- *Operational natural faults (issue 4):* faults caused by a natural phenomenon without human participation, affecting hardware as well as information stored on hardware and occurring during the service of the system. As they affect hardware faults are likely to propagate to software as well. They can be, for instance, a memory alteration due to a cosmic radiation.
- *Operational human-errors (issue 5):* faults resulting from human action during the use of the system. They include faults affecting the hardware and the software, being deliberate or non-deliberate but don't encompass malicious ones. Connection between this taxonomy and classical human error classification as the one defined in [80] can be easily made with deliberate faults corresponding to mistakes or violations [76] and non-deliberate ones being either slips or lapses. [37] describes precisely how these errors can be connected to the description of operators' work in task models.

3.2 Faults Altering the Human

The classification presented in this section expands Avizienis' taxonomy in four ways. First, we extend the *System boundary* dimension to recognize that human faults can be **induced inside the operator** from external causes. Second, we add new levels to the *Phenomenological cause* dimension to distinguish between faults arising 1) from the operator, 2) from another person, and 3) from the natural world (including the system itself). Third, we introduce the *Human capability* dimension to differentiate faults in the operator's perceptual, cognitive, and motor abilities. Fourth, we add specific fault categories that derive from these dimensions. This presentation

In particular, the complex interactions between an operator and a system (following the architecture presented in **Fig. 3**) have properties and characteristics that are separate from the operator alone or the system alone, and the architecture can lead to many different types of faults that have many different underlying causes – some of which involve the fault being "induced" in the operator by outside forces. For example, an aircraft's hard landing may arise from within the operator (e.g., a pilot's early-stage Parkinson's disease that reduces their muscular coordination), from another person (e.g., someone shining a laser pointer into the pilot's eyes from the end of the runway), or from effects of the natural world (e.g., air turbulence that shakes a pilot's arm as they try to press a button on the instrument panel). Although these three faults are very different in terms of implications for design, they would all be placed in the same category in the Avizienis framework (i.e., "Operational / External / Human-made / Non-malicious / Non-deliberate / Accidental" operator faults). To address this gap, we need to broaden the dimensions that characterize faults. The classification presented in [71] focusses only on operational faults (leaving aside the development faults and their causes but their types are similar [83]).

The classification expands the *System boundary* dimension to add the architecture of **Fig. 3** as a conceptual location for faults that should be considered separately from Avizienis et al.'s categories of "internal to the system" and "external to the system."

The idea of internal/external faults separation applied to the architecture of **Fig. 3** separates faults that arise from inside the operator (see **Fig. 5** bottom) and those that arise external to the operator (see **Fig. 5** top).

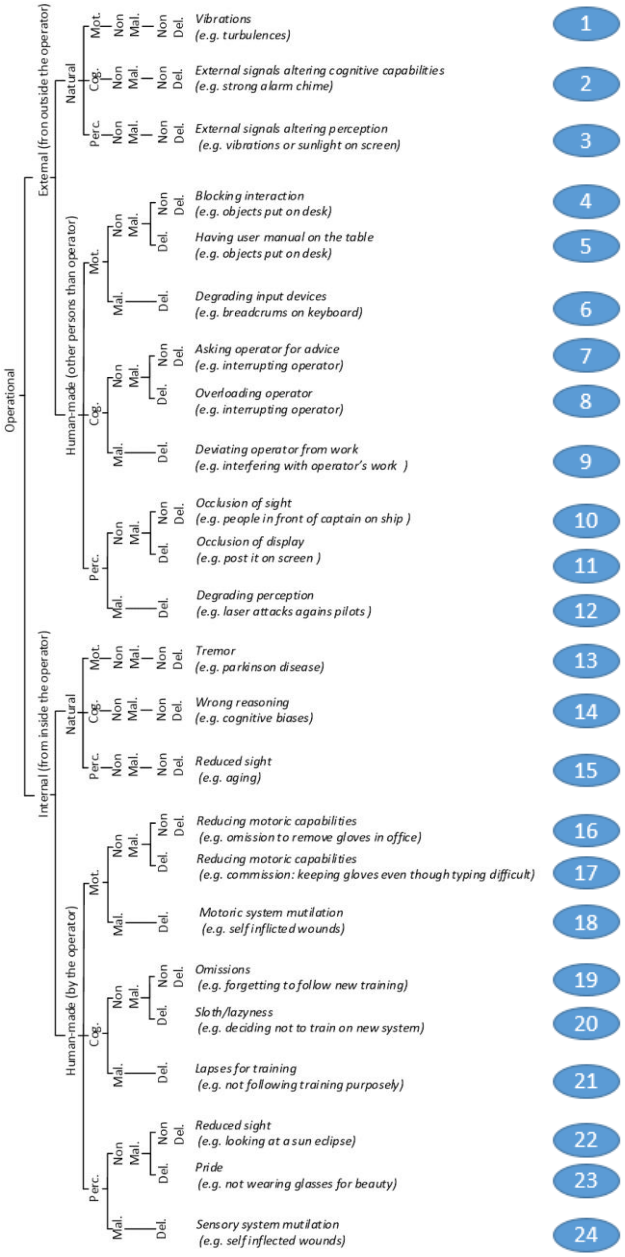

Fig. 5. The taxonomy on Internal and External faults altering the capability of the operator as a service provider (with concrete examples – right-hand side in italics)

This classification covers various types of influential factors for people behaviors such as seven deadly sins [31] (e.g. items 20 and 23 in **Fig. 5**), cognitive biases [89] (e.g. item 14), aging [92] (e.g. items 13 and 15) as well as more standard human error classification [80] (e.g. items 19 and 21).

4 The Quest for Perfect Software

As other types of computing systems [16], interactive system development follows the three basic principles of incentives in economy: Economic Incentives, Social incentives and Moral incentives (as highlighted in [58]). Economic incentives concern the real development costs and, for instance, detailed information about usability evaluation costs can be found in [15]. Beyond, low quality software exposes software developers and distributors to legal risks [97] that contribute as an economic incentive. Moral incentive will motivate designers and developer to follow their "moral compass" which could prevent them from performing low quality work due, for instance, to laziness [6]. Last, social incentives [36], could be used to develop people's natural desire to be looked upon favorably by others. On the flip side, people fear being shamed and looked upon disfavorably by their peers. This means that control quality and assessment of quality of production might incent them to produce artefacts of better quality.

These three incentives have a strong influence on developers and designers' behavior and might, if well exploited, contribute to the development of interactive systems of better quality. Unfortunately, they are also conflicting as, for instance, increase in quality assessment will increase the development cost and thus reduce the economic incentive.

4.1 Expected Properties of Interactive Software

With the early work on understanding interactive systems [33] came the identification of properties that "good" interactive systems should exhibit (e.g. honesty) and "bad" properties that they should avoid (e.g. deadlocks). Later, guidelines for the design of interactive systems [95] were provided, identifying in a similar way "good" properties (e.g. guidance), in order to favor usability of these systems. In the area of software engineering, early work [55] identified two main good properties of computing systems namely safety (i.e. nothing bad will ever happen) and liveness (i.e. something good will eventually happen). In [60] a hierarchy of software properties is proposed identifying for the first time explicit relationships between properties gathered in a hierarchy (e.g. "reactivity" divided in "recurrence" and "persistence"). While in the area of Human-Computer Interaction the properties were initially expressed in an informal way, [75], [74] proposed the use of temporal logics to describe these properties.

Beyond these "generic" properties, it is of interest to represent specific properties related to the very nature of each system. These properties might also be of a high level of abstraction (e.g. trust for a banking system) or of very low level (e.g. only possible to enter a personal identification number three times on a cash machine). The detailed property would contribute to the high-level one.

Usability and User Experience. These two major properties in Human-Computer Interaction do not have currently the same level of maturity. Usability has been studied since the early 80's and has been standardized by ISO in the ISO 9241 part 11 since 1996 [50]. Its structure is presented on the a) section of **Fig. 6**. The standard specializes Usability into three sub-properties (efficiency, effectiveness and satisfaction) while some researchers would also add at least Learnability and Accessibility [47] as important aspects of Usability.

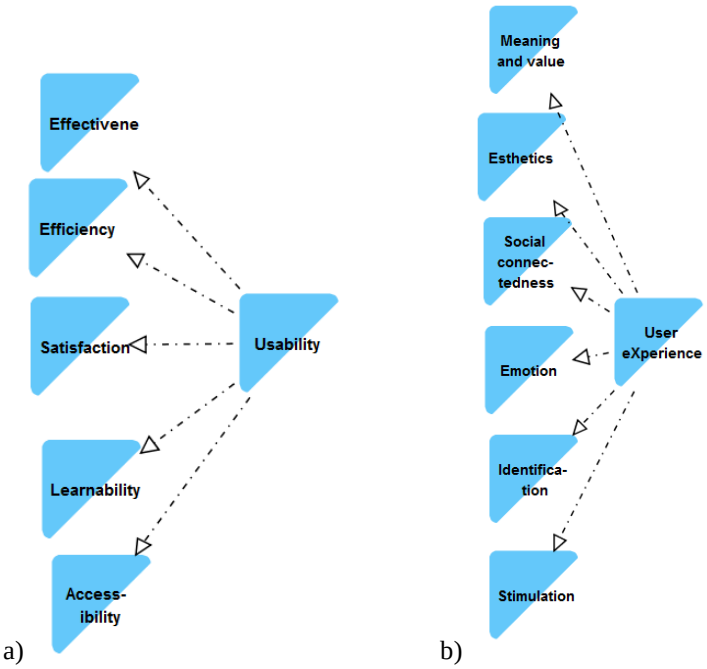

Fig. 6. Representation of the hierarchical relationships between properties and their contributing factors for a) Usability [50, 68] and b) User Experience [73]

User Experience is a more recent concept that is under standardization but still not mature. Sub-properties of User Experience (usually called dimensions) are diverse in terms of level of abstraction and vary widely amongst authors (see [47] for a description of user experience in terms of hedonic and ergonomic qualities – another word for properties). [73] proposes the only set of dimensions that has been carefully check for orthogonality and proposes six dimensions at the same level of abstraction (see right-hand side b) section of **Fig. 6**)

Dependable and Secure Computing and Concurrent Programs Properties. The first issue of the IEEE transactions on Dependable and secure computing included a paper [7] dedicated to a taxonomy of properties of those systems. The taxonomy is presented in part a) of **Fig. 7**. Beyond a very clear definition of each property this classification shows that some sub-properties such as availability are related to higher-level

properties namely safety and security. Indeed, a loss of availability might impact dependability of the systems (if the service not available is requested) while security attacks might target at a reduction of availability of service (as in the classical DDoS – Distributed Denial of Service).

The right-hand side of **Fig. 7** presents a very old and classical decomposition of properties of concurrent systems: safety and liveness that have been introduced in the introduction. Beyond this separation, Sistla proposed in [84] a refinement of these properties in more precise ones contributing to the presence or the absence of the more high-level ones.

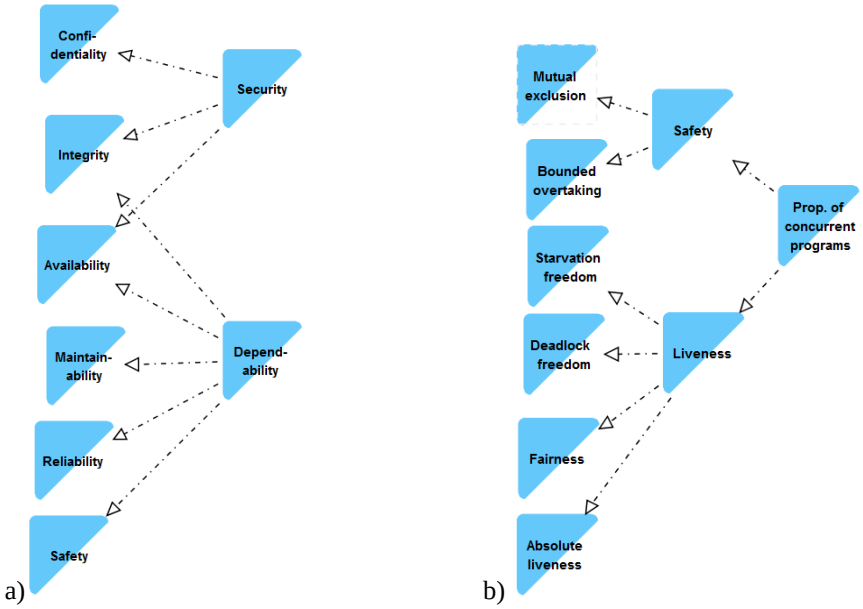

Fig. 7. Representation of hierarchical relationships between properties and contributing factors for Security and Dependability [7] (a) and for concurrent programs [75, 74]

A more comprehensive description of hierarchies of properties for interactive systems and a dedicated notation to represent them and their possible conflicts can be found in [39].

4.2 Processes Supporting the Presence of Expected Properties

User Centered Design Processes. These processes as defined in [51] promotes taking into account usability (especially satisfaction and efficiency) and user experience by iterative processes involving explicitly real users in the design and evaluation phases [45]. Effectiveness is addressed by processes promoting explicit description of user work and tasks as in Cognitive Work Analysis [96] or task-centered processes [62]. While focusing on these "user-centered" properties, these approaches tend to lower the importance of the "system-centered" other properties as this is the case for agile pro-

cesses focusing of early delivery of low quality systems [93]. According to the classification of faults in **Fig. 4** by supporting usability, these processes would support addressing human-made development faults (by having usable Integrated Development Environments) as well as human-made operational faults (by designing usable interactive applications).

Dependability Centered Design Processes. In the field of critical systems, safety standards such as DO-178C or IEC 61508 define Development Assurance Levels for software systems (or for functions of software systems). These levels are based on the analysis of consequences or effect of a malfunction. For instance, if a function failure has high consequences such as multiple fatalities, it is called catastrophic and certification authorities will require that the system manufacturer will provide a Development Assurance Level A (DO-178C standard for aeronautics [34]). If consequences are lower, the required level will decrease. Developing a system of a DAL A is extremely resource consuming and expensive and, as far as software is concerned, the use of formal description techniques is required [35]. In lower DALs, such expensive approaches are not required and for reaching levels such as DAL D rigorous software engineering approaches are sufficient.

According to the classification of faults in **Fig. 4** by supporting reliability, these processes would support addressing human-made development faults (by using formal methods to detect defects in the code).

Processes integrating dependability, usability and user experience. Some processes (such as [11]) which focusses on the use of formal models of the interactive system to assess usability (mainly efficiency and satisfaction) and [61] which focusses on the effectiveness dimension of usability (see decomposition of usability in **Fig. 6**). Merging these two approaches is very difficult and few contributions address it [12]. Indeed, this brings specific issues such as the expertise of developers and designers but also the economic benefits in the case of non-critical interactive applications.

5 Techniques and Approaches for Addressing Faults

This section presents several fault-tolerant mechanisms designed in several research domains such as dependable computing, formal methods and human-computer interaction providing means to avoid, detect, remove, tolerate or mitigate the faults presented in **Fig. 4** and **Fig. 5**.

5.1 Techniques for Addressing Human Faults

Heuristic evaluations. The ten heuristics from [68] aims at support experts in detecting defects in an interactive application and avoiding the operational human-made faults from **Fig. 4**. In can also support detecting some of the faults affecting the operator in **Fig. 5** but more the human-made ones (bottom of the figure) rather than the other ones.

UCD processes. As explained above, techniques deployed in UCD processes aim at detecting development faults (at design level) by involving users through user testing. Beyond detection, processes focusing on creativity as [21] aim at identifying solutions that would remove the fault by proposing better designs in terms of usability and user experience.

Debiasing cognitive biases. Several hundred of cognitive biases have been identified in the literature. The cognitive biases codex [89] breaks down cognitive errors into four quadrants: memory, meaning, information overload, and need for speed . Others [10] have proposed different grouping according to the general mental problem they attempt to address: too much information, not enough meaning, need to act fast, what should be remembered. In the field of HCI some specific biases have been studied (e.g. peak-end effect [27]) and their use for design (e.g. organizing work over multiple pages taking into account peak-end effect) has been proposed. Similarly, work from Saint-Lot et al. [81] proposes a graphical countermeasure to cognitive tunneling bias (an orange-red flash of 300 milliseconds with a 15% opacity) to improve reaction time of air traffic controller and mitigate attention tunneling bias. However, such research contributions are not connected with each other and propose local solution to selected cognitive biases (on a one by one basis).

Environment disturbance tolerance. The environment in which the system is deployed can deeply degrade operators' performance. This is because the environment triggers faults on the operator without possibly affecting the interactive system. This type of fault is represented, for instance, in item 1 of **Fig. 5**. The turbulences trigger a natural fault that sets the operator in an error mode [70]. If the operator needs to provide input to the interactive system (in that case an aircraft cockpit) the likelihood of error is very high. To prevent such error a new interaction technique called "brace touch" has been proposed that nearly remove all the operators' errors in case of light and severe turbulences [26].

User interface services. Specific function such as copy-paste or undo are added to user interfaces in order to prevent operational faults such as triggering a command inadvertently or making mistake or slips (as defined in [80]) when type the same text in another place. These faults may occur at development and operational times demonstrating the need to encompass these services both in IDEs and in interactive applications.

5.2 Techniques for Addressing Interactive System Faults

Self-checking software components (redundancy, diversity and segregation). As introduced in [7] and [57], many dependability strategies rely on replicated self-checking components as they provide error-confinement and can thus be considered as fail-stop components. The COM/MON approach [91] is the basis for various N-Self-Check-

ing Programming (NSCP)-based architectures [57]. A self-checking software component can be roughly described as composed of two pieces of software, the first one (functional component) being the classical component and the second one (monitoring component) being in charge of checking its execution and outputs and being able to send error notifications in case of inconsistency. As both pieces receive the same input, fault (e.g. natural faults in **Fig. 4**) will be detected if both pieces produce inconsistent output. In order to correct the natural faults, more dissimilar (but functionally equivalent) pieces have to be executed in parallel. Having a voting mechanism checking the output of the pieces will allow detecting a fault (if they don't provide the same output) but also remove the fault (by following the majority of outputs that are the same). Self-cheking approach (embedding redundancy, diversity and segregation) have been applied to interactive systems in the area of aircraft cockpits [86].

Formal verification. Formal verification aims at exploiting mathematical reasoning over a model (or directly the code) of the system to detect defects (corresponding usually to properties not being true). It thus aims at detecting development faults and usually also provide means or support to remove them. For instance, model checking tools will verify is a property is true on a specification and if not will provide a counter example (a sequence of actions that lead to a state where the property does not hold) [53]. These approaches have been applied for many years to interactive systems, starting from WIMP interaction technique [66] to more sophisticated ones such as multitouch [46] or even brace touch introduced in the previous section [70]. In order to ensure that the verification is performed correctly, tools and tool suites are developed. [28] proposes a systematic comparison of formal tools for interactive systems highlighting both the benefits of these approaches and their limitations. User-related properties can also be checked, at least partly, using formal verification techniques as demonstrated in [72] which demonstrate how to verify ergonomic rules over a formal model of an interactive, post-WIMP, application.

Interactive software testing. Software testing is another type of technique for detecting development faults. The basic idea is to run a large number of test cases in order to detect behaviors that are incompatible with the requirements. Software testing has been developed for many years and, in order to deal with the complexity of the cases, model-based testing is nowadays the most prominent approach [94]. As for model-checking, software testing support also the identification of defects and debugging [65]. In the area of interactive systems, testing is a complex tasks as user actions are unpredictable and the number of cases is infinite [20]. Formal model-based approaches offering formal verification have recently been combined to detect and correct defects in interactive systems encompassing both hardware and software aspects of the architecture in **Fig. 3** [18].

6 Conclusion and Perspectives

This paper has presented a generic framework called POISE (People, Organizations, Interactive Systems and Environment) that presents in an integrated way four different aspects affecting deeply interactive systems development and exploitation. Based on two complementary on faults taxonomy we have presented a comprehensive coverage of faults altering the functioning of interactive systems (i.e. human-made and natural faults) and the behavior of people (i.e. internal and external to the operator) including cognitive biases, deadly sins and standards operators.

Beyond, the paper has offered an overview of processes, methods and techniques offering various means to address all these faults. These contributions come from different research domains such as Formal Methods, Dependable Computing and Software Engineering and Human-Computer Interaction. While they usually try to tackle a single type of faults, some (which incorporate techniques stemming from multiple domains) have been trying to provide more complex solutions dedicated to the design, development, evaluation and the operation of interactive system embedding multiple (sometimes conflicting) properties such as usability, user experience, reliability, dependability, security, safety … among many other ones [5].

Due to space constraints and for keeping the message simple, the issues related to organizational aspects are not presented. It is however at least as important as the other three aspects as the organization structure the work of operators, define the requirements, select the development processes and techniques to be used by designers and developer and above all plan training and organize operators in teams. Incidents and accidents stemming from the organizations are numerous as demonstrated in [78] as they might jeopardize all the efforts on the other aspects. As stated in the introduction, even though the people are identified as sources of faults, we would advocate that designs should not aim at removing them even though this is the path promoted by Artificial Intelligence (targeting at unmanned systems as drones or so-called autonomous vehicles produced with as limited as possible human intervention. On the opposite, design and development should rely on trained and qualified operators, supported by usable and efficient tools in order to ensure that the human contribution will be fully present in future interactive systems. Knowledge, tools and empowerment of designers and developers is the only path to deploying perfect interactive systems.

Acknowledgements. The author would like to acknowledge support from the ICS team in Toulouse (E. Barboni, D. Navarre and C. Martinie) for working on most of the contributions presented. Special also goes to Yannick Deleris and Christine Gris from Airbus who contributed and funded part of this work.

References

1. Accot J.and Zhai S. 2003. Refining Fitts' law models for bivariate pointing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (*CHI '03*). Association

for Computing Machinery, New York, NY, USA, 193–200. DOI:https://doi.org/10.1145/642611.642646

2. Albinsson, P.A. & Zhai, S. (2003) High Precision Touch Screen Interaction. Proc. ACM CHI conference, pp. 105-112
3. Android Material Design guidelines https://material.io/design/guidelines-overview (retrieved July 6th 2021)
4. Antti Kangasrääsiö, Kumaripaba Athukorala, Andrew Howes, Jukka Corander, Samuel Kaski, and Antti *Oulasvirta*. 2017. Inferring Cognitive Models from Data using Approximate Bayesian Computation. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1295–1306. DOI:https://doi.org/10.1145/3025453.3025576
5. Ardito C., Bernhaupt R., Palanque P., Sauer S. (2019) Handling Security, Usability, User Experience and Reliability in User-Centered Development Processes. In: Lamas D., Loizides F., Nacke L., Petrie H., Winckler M., Zaphiris P. (eds) Human-Computer Interaction – INTERACT 2019. INTERACT 2019. Lecture Notes in Computer Science, vol 11749. Springer, Cham. https://doi.org/10.1007/978-3-030-29390-1_76
6. Armour P. G. The business of software estimation is not evil. Communications of the ACM, 57(1):42–43, 2014.
7. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing, vol.1, no.1, pp. 11- 33, Jan.-March 2004
8. Beaudouin-Lafon M. 2004. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces* (*AVI '04*). Association for Computing Machinery, New York, NY, USA, 15–22. DOI:https://doi.org/10.1145/989863.989865
9. Beirekdar A., Keita M., Noirhomme M., Randolet F., Vanderdonckt J., Mariage C. (2005) Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites. In: Costabile M.F., Paternò F. (eds) Human-Computer Interaction - INTERACT 2005. INTERACT 2005. Lecture Notes in Computer Science, vol 3585. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11555261_25
10. Benson B. 2016. Cognitive biases cheat sheet. https://medium.com/better-humans/cognitive-bias-cheat-sheet-55a472476b18 (retrieved July 2021).
11. Bernhaupt R., Navarre D., Palanque P., Winckler M. (2008) Model-Based Evaluation: A New Way to Support Usability Evaluation of Multimodal Interactive Applications. In: Law E.LC., Hvannberg E.T., Cockton G. (eds) Maturing Usability. Human-Computer Interaction Series. Springer, London. https://doi.org/10.1007/978-1-84628-941-5_5
12. Bernhaupt R., Palanque P., Manciet F., Martinie C. (2016) User-Test Results Injection into Task-Based Design Process for the Assessment and Improvement of Both Usability and User Experience. In: Bogdan C. et al. (eds) Human-Centered and Error-Resilient Systems Development. HESSD 2016, HCSE 2016. Lecture Notes in Computer Science, vol 9856. Springer, Cham. https://doi.org/10.1007/978-3-319-44902-9_5
13. Berry R. Common User Access – A consistent and usable human-computer interface for the SAA environments". IBM Systems Journal, Volume 27, Nº 3, 1988
14. Bev Littlewood and Lorenzo Strigini. 2000. Software reliability and dependability: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (*ICSE '00*). Association for Computing Machinery, New York, NY, USA, 175–188. DOI:https://doi.org/10.1145/336512.336551
15. Bias R. G. and Mayhew D. 2005. *Cost-Justifying Usability: An Update for the Internet Age*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

16. Boehm B. and Sullivan K. 2000. Software economics: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (*ICSE '00*). Association for Computing Machinery, New York, NY, USA, 319–343. DOI:https://doi.org/10.1145/336512.336584
17. Bouzekri E., Canny A., Martinie C., Palanque P., Gris C. (2019) Deep System Knowledge Required: Revisiting UCD Contribution in the Design of Complex Command and Control Systems. In: Lamas D., Loizides F., Nacke L., Petrie H., Winckler M., Zaphiris P. (eds) Human-Computer Interaction – INTERACT 2019. INTERACT 2019. Lecture Notes in Computer Science, vol 11746. Springer, Cham. https://doi.org/10.1007/978-3-030-29381-9_42
18. Bouzekri E., Martinie C., Palanque P., Atwood K., Gris C. Should I add Recommendations to my Warning System? The RCRAFT Framework can Answer This and Other Questions about Supporting the Assessment of Automation Designs. IFIP TC 13 INTERACT 2021 conference, LNCS, Springer.
19. Bowen J. and Stavridou V. Formal Methods, Safety-Critical Systems and Standards. Software Engineering Journal, 8(4):189–209, July 1993
20. Bowen J.and Reeves S. 2017. Generating Obligations, Assertions and Tests from UI Models. *Proc. ACM Hum.-Comput. Interact.* 1, EICS, Article 5 (June 2017), 18 pages. DOI:https://doi.org/10.1145/3095807
21. Buxton B. 2007. Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
22. Canny A., Martinie C., Navarre D., Palanque P., Barboni E., and Gris C. 2021. Engineering Model-Based Software Testing of WIMP Interactive Applications: A Process based on Formal Models and the SQUAMATA Tool. *Proc. ACM Hum.-Comput. Interact.* 5, EICS, Article 207 (June 2021), 30 pages. DOI:https://doi.org/10.1145/3461729
23. Card S., Moran T., Newell A. The psychology of human-computer interaction. Erlbaum 1983, ISBN 0898598591, pp. I-XIII, 1-469
24. Card, S.K; Moran, T. P; and Newell, A. The Model Human Processor: An Engineering Model of Human Performance. Handbook of Perception and Human Performance. Vol. 2: Cognitive Processes and Performance, 1986, pages 1–35.
25. Cockburn A., Gutwin C., Palanque P., Deleris Y., Trask C., Coveney A., Yung M., and MacLean K. 2017. Turbulent Touch: Touchscreen Input for Cockpit Flight Displays. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 6742–6753. DOI:https://doi.org/10.1145/3025453.3025584
26. Cockburn A., Masson D., Gutwin C., Palanque P., Goguey A., Yung M., Gris C., Trask C. Design and evaluation of braced touch for touchscreen input stabilisation, International Journal of Human-Computer Studies, Volume 122, 2019, Pages 21-37, ISSN 1071-5819, https://doi.org/10.1016/j.ijhcs.2018.08.005
27. Cockburn A., Quinn P., and Gutwin C. 2015. Examining the Peak-End Effects of Subjective Experience. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (*CHI '15*). Association for Computing Machinery, New York, NY, USA, 357–366. DOI:https://doi.org/10.1145/2702123.2702139
28. Creissac Campos J., Fayollas C., Harrison M. D., Martinie C., Masci P., and Palanque P. 2020. Supporting the Analysis of Safety Critical User Interfaces: An Exploration of Three Formal Tools. *ACM Trans. Comput.-Hum. Interact.* 27, 5, Article 35 (October 2020), 48 pages. DOI:https://doi.org/10.1145/3404199
29. Cronel M., Dumas B., Palanque P., Canny A. (2019) MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In Human-Centered Software Engineering. HCSE 2018. LNCS, vol 11262. Springer.

30. Cronel M., Dumas B., Palanque P., Canny A. (2019) MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In: Bogdan C., Kuusinen K., Lárusdóttir M., Palanque P., Winckler M. (eds) Human-Centered Software Engineering. HCSE 2018. Lecture Notes in Computer Science, vol 11262. Springer
31. Diamantaris M., Marcantoni F., Ioannidis S., and Polakis J. 2020. The Seven Deadly Sins of the HTML5 WebAPI: A Large-scale Study on the Risks of Mobile Sensor-based Attacks. *ACM Trans. Priv. Secur.* 23, 4, Article 19 (August 2020), 31 pages. DOI:https://doi.org/10.1145/3403947
32. Diaper D. & Stanton N. The handbook of task analysis for human-computer interaction. Lawrence Erlbaum Associates, 2003. ISBN 0-8058-4432-5
33. Dix A. Abstract, Generic Models of Interactive Systems. BCS HCI conference 1988, 63-77 (1988).
34. DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification, published by RTCA and EUROCAE, 2012.
35. DO-333 Formal Methods Supplement to DO-178C and DO-278A, published by RTCA and EUROCAE December 13, 2011.
36. Eisenberg N. and Miller P. A. 1987. The relation of empathy to prosocial and related behaviors. Psychological Bulletin 101, 1 (1987), 91
37. Fahssi R., Martinie C., Palanque P. (2015) Enhanced Task Modelling for Systematic Identification and Explicit Representation of Human Errors. In: Abascal J., Barbosa S., Fetter M., Gross T., Palanque P., Winckler M. (eds) Human-Computer Interaction – INTERACT 2015. INTERACT 2015. Lecture Notes in Computer Science, vol 9299. Springer, Cham. https://doi.org/10.1007/978-3-319-22723-8_16
38. Fayollas C., Fabre J. C., Palanque P., M. Cronel, D. Navarre and Y. Deleris, "A Software-Implemented Fault-Tolerance Approach for Control and Display Systems in Avionics," *2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing*, 2014, pp. 21-30, doi: 10.1109/PRDC.2014.11.
39. Fayollas C., Martinie C., Palanque P., Ait-Ameur Y., FORMEDICIS (2018) QBP Notation for Explicit Representation of Properties, Their Refinement and Their Potential Conflicts: Application to Interactive Systems. In: Clemmensen T., Rajamanickam V., Dannenmann P., Petrie H., Winckler M. (eds) Global Thoughts, Local Designs. INTERACT 2017. Lecture Notes in Computer Science, vol 10774. Springer, Cham. https://doi.org/10.1007/978-3-319-92081-8_9
40. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis & design language (AADL): An introduction (No. CMU/SEI-2006-TN-011). CMU Software Engineering Inst (2006)
41. Feyisetan O. and Simperl E.. 2017. Social Incentives in Paid Collaborative Crowdsourcing. *ACM Trans. Intell. Syst. Technol.* 8, 6, Article 73 (September 2017), 31 pages. DOI:https://doi.org/10.1145/3078852
42. Feyisetan O. and Simperl E.. 2019. Beyond Monetary Incentives: Experiments in Paid Microtask Contests. *Trans. Soc. Comput.* 2, 2, Article 6 (October 2019), 31 pages. DOI:https://doi.org/10.1145/3321700
43. Fitt, P. M. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology. 1954; 47:pp. 381-391.
44. Flemisch, F., Heesen, M., Hesse, T. et al. Towards a dynamic balance between humans and automation: authority, ability, responsibility and control in shared and cooperative control situations. Cogn Tech Work 14, 3–18 (2012). https://doi.org/10.1007/s10111-011-0191-6
45. Gould, I.D., and Lewis, C. Designing for usability: Key principles and what designers think. Commun. ACM 28, 3 (Mar. 1985), 300-311.

46. Hamon A., Palanque P., Silva J-L., Deleris Y., and Barboni E. 2013. Formal description of multi-touch interactions. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems* (*EICS '13*). Association for Computing Machinery, New York, NY, USA, 207–216. DOI:https://doi.org/10.1145/2494603.2480311
47. Hassenzahl M., Platz A., Burmester M., Lehner K. Hedonic and ergonomic quality aspects determine a software's appeal. ACM CHI conference 2000: ACM DL, 201-208
48. Hecht H. and Fiorentino E. Reliability assessment of spacecraft electronics. In Annual Reliability and Maintainability Symp., pages 341–346. IEEE, 1987
49. Hollnagel E. How Resilient Is Your Organisation? An Introduction to the Resilience Analysis Grid (RAG). Sustainable Transformation: Building a Resilient Organization, May 2010,Toronto, Canada
50. International Standard Organization: "ISO 9241-11." Ergonomic requirements for office work with visual display terminals (VDT) – Part 11 Guidance on Usability (1996).
51. ISO 9241-210: Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems, Geneva
52. Johnson P. 1992. Human-Computer Interaction: psychology, task analysis and software engineering, McGraw Hill, Maidenhead, UK
53. Kupferman, O., Vardi, M.Y. Model Checking of Safety Properties. *Formal Methods in System Design* **19,** 291–314 (2001). https://doi.org/10.1023/A:1011254632723
54. Lai-Chong, E., Roto, V., Hassenzahl, M., Vermeeren, A.: Kort. J. Understanding, scoping and defining user experience: a survey approach. In: Proceedings of the 27th International Conference on Human Factors in Computing Systems, pp. 719–728. ACM, NY (2009)
55. Lamport, L.: Proving the correctness of multiprocess programs. IEEE transactions on software engineering (2), 125-143 (1977)
56. Landwehr C., Bull A., McDermott J., and Choi W. 1994. A taxonomy of computer program security flaws. *ACM Comput. Surv.* 26, 3 (Sept. 1994), 211–254. DOI:https://doi.org/10.1145/185403.185412
57. Laprie, J-C. et al. Definition and analysis of hardware and software fault-tolerant architectures, IEEE Computer, Vol. 23, No. 7, pp.39–51. 1990
58. Levitt, S. D., & Dubner, S. J. (2005). Freakonomics: A rogue economist explores the hidden side of everything. New York: William Morrow.
59. Littlewood B. and Strigini L. 2000. Software reliability and dependability: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (*ICSE '00*). Association for Computing Machinery, New York, NY, USA, 175–188. DOI:https://doi.org/10.1145/336512.336551
60. Manna, Z., Pnueli, A.: A Hierarchy of Temporal Properties. ACM Symposium on Principles of Distributed Computing1990: 377-410 (1990).
61. Martinie C., Navarre D., Palanque P., and Fayollas C. 2015. A generic tool-supported framework for coupling task models and interactive applications. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (*EICS '15*). ACM, 244–253. DOI:https://doi.org/10.1145/2774225.2774845
62. Martinie C., Palanque P., Navarre D., Barboni E. (2012) A Development Process for Usable Large Scale Interactive Critical Systems: Application to Satellite Ground Segments. In: Winckler M., Forbrig P., Bernhaupt R. (eds) Human-Centered Software Engineering. HCSE 2012. Lecture Notes in Computer Science, vol 7623. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34347-6_5

63. Maryam Mustafa, Lea Lindemann, and Marcus Magnor. 2012. EEG analysis of implicit human visual perception. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 513–516. DOI:https://doi.org/10.1145/2207676.2207746
64. Meshkati, N. (1989). Technology transfer to developing countries: a tripartite micro- and macro ergonomic analysis of human-organization-technology interfaces. International Journal of Industrial Ergonomics, 4, 101-115
65. Navabpour S., Bonakdarpour B., and Fischmeister S. 2011. Software debugging and testing using the abstract diagnosis theory. In *Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems* (*LCTES '11*). Association for Computing Machinery, New York, NY, USA, 111–120. DOI:https://doi.org/10.1145/1967677.1967693
66. Navarre D., Palanque P., Bastide R., Sy O. (2001) Structuring Interactive Systems Specifications for Executability and Prototypability. In: Palanque P., Paternò F. (eds) Interactive Systems Design, Specification, and Verification. DSV-IS 2000. Lecture Notes in Computer Science, vol 1946. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44675-3_7
67. Nicolescu B., Peronnard P., Velazco R., and Savaria Y. Efficiency of Transient Bit-Flips Detection by Software Means: A Complete Study. Proc. of the 18th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT '03). IEEE Computer Society, 377-384
68. Nielsen J. 1994. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
69. Olwal, A. and Feiner, S. Rubbing the Fisheye: Precise Touch-Screen Interaction with Gestures and Fisheye Views. Conference Supplement of UIST 2003. pp. 83-84.
70. Palanque P., Cockburn A., Désert-Legendre L., Gutwin C., Deleris Y. (2019) Brace Touch: A Dependable, Turbulence-Tolerant, Multi-touch Interaction Technique for Interactive Cockpits. In: Romanovsky A., Troubitsyna E., Bitsch F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2019. Lecture Notes in Computer Science, vol 11698. Springer, Cham. https://doi.org/10.1007/978-3-030-26601-1_4
71. Palanque P., Cockburn A., Gutwin C. (2020) A Classification of Faults Covering the Human-Computer Interaction Loop. In: Casimiro A., Ortmeier F., Bitsch F., Ferreira P. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2020. Lecture Notes in Computer Science, vol 12234. Springer, Cham. https://doi.org/10.1007/978-3-030-54549-9_29
72. Palanque, P., Farenc, Ch., & Bastide, R. Embedding Ergonomic Rules as Generic Requirements in a Formal Development Process of Interactive Software. In Proceedings of IFIP TC 13 Conference on Human-Computer Interaction INTERACT'99 (Edinburg, Scotland, 1-4 September 1999.
73. Pirker, M. and Bernhaupt, R.: Measuring user experience in the living room: results from an ethnographically oriented field study indicating major evaluation factors. EuroITV 2011, 79-82 (2011).
74. Pnueli A.: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. LNCS n° 224 p.510-584. Springer Verlag (1986).
75. Pnueli, A.: The Temporal Logic of Programs. 18th IEEE symposium on the Foundations of Computer Science, 46-57 (1977)
76. Polet, P, Vanderhaegen, F, and Wieringa, P. Theory of safety related violation of system barriers. Cognition Technology & Work, 4, 3, 171-179. 2002.
77. Raber F., Kosmalla F., Krueger A. (2017) Fine-Grained Privacy Setting Prediction Using a Privacy Attitude Questionnaire and Machine Learning. In: Bernhaupt R., Dalvi G., Joshi A., K. Balkrishan D., O'Neill J., Winckler M. (eds) Human-Computer Interaction – INTERACT

2017. INTERACT 2017. Lecture Notes in Computer Science, vol 10516. Springer, Cham. https://doi.org/10.1007/978-3-319-68059-0_48
78. Reason J. Managing the Risks of Organizational Accidents. Ashgate Publishing limited. 1997.
79. Reason J. The Human Contribution: Unsafe acts, Accidents and Heroic Recoveries. Routeledge, Taylor and Francis, 2008.
80. Reason, J. (1990). Human Error, Cambridge University Press
81. Saint-Lot J., Imbert J-P. and Dehais F. 2020. Red Alert: A Cognitive Countermeasure to Mitigate Attentional Tunneling. In *Proceedings of CHI '20: CHI Conference on Human Factors in Computing Systems (CHI '20), April 25-30, 2020, Honolulu, HI, USA*. ACM, New York, NY, USA, https://doi.org/10.1145/3313831.3376709
82. Schroeder B., E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: a large-scale field study. In ACM SIGMETRICS, pages 193–204, Seattle, WA, June 2009.
83. Shah P., Berges M., and Hubwieser P. 2017. Qualitative Content Analysis of Programming Errors. In *Proceedings of the 5th International Conference on Information and Education Technology* (*ICIET '17*). ACM, 161–166. DOI:https://doi.org/10.1145/3029387.3029399
84. Sistla, A. P. On characterization of safety and liveness properties in temporal logic. In: Proceedings of the fourth annual ACM symposium on Principles of distributed computing, pp. 39-48, ACM (1985).
85. Soukoreff W. & MacKenzie S. Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. IJHCS. 61(6): 751-789 (2004)
86. Tankeu Choitat A, Fabre J.-C., Palanque P., Navarre D., and Deleris Y. 2011. Self-checking widgets for interactive cockpits. In *Proceedings of the 13th European Workshop on Dependable Computing* (*EWDC '11*). Association for Computing Machinery, New York, NY, USA, 43–48. DOI:https://doi.org/10.1145/1978582.1978592
87. Tankeu-Choitat A., Navarre D., Palanque P., Deleris Y., Fabre J-C. and Fayollas C. Self-Checking Components for Dependable Interactive Cockpits Using Formal Description Techniques, *2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing*, 2011, pp. 164-173, doi: 10.1109/PRDC.2011.28.
88. ter Beek M.H., Faconti G.P., Massink M., Palanque P.A., Winckler M. (2009) Resilience of Interaction Techniques to Interrupts: A Formal Model-Based Approach. In: Gross T. et al. (eds) Human-Computer Interaction – INTERACT 2009. INTERACT 2009. Lecture Notes in Computer Science, vol 5726. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-03655-2_56
89. The cognitive biases codex: 175 cognitive biases https://medium.com/better-humans/cognitive-bias-cheat-sheet-55a472476b18 (retrieved July 8th 2021)
90. Thoma V., White E.P. (2011) In Two Minds about Usability? Rationality and Intuition in Usability Evaluations. In: Campos P., Graham N., Jorge J., Nunes N., Palanque P., Winckler M. (eds) Human-Computer Interaction – INTERACT 2011. INTERACT 2011. Lecture Notes in Computer Science, vol 6949. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23768-3_78
91. Traverse, P., Lacaze, I. and Souyris, J. Airbus fly-by-wire: a total approach to dependability, Proc. WCC, pp.191–212. 2004
92. Trewin S., John B., Richards J., Sloan D., Hanson V., Bellamy R., Thomas J. and Swart C.. 2012. Age-specific predictive models of human performance. *CHI '12 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2267–2272. DOI:https://doi.org/10.1145/2212776.2223787
93. Turk, D., France, R., Rumpe, B. Limitations of agile software processes. In: Proc.Int. Conf. on eXtreme Programming and Agile Processes in Software Engineering Italy (2002).

94. Utting M., Pretschner A., and Legeard B. 2012. A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22, 5 (August 2012), 297--312. DOI:https://doi.org/10.1002/stvr.456
95. Vanderdonckt, J.: Development milestones towards a tool for working with guidelines. Interacting with Computers 12(2), 81-118 (1999).
96. Vicente K. 1999. Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work. Lawrence Erlbaum Associates
97. Wahl N. J.. 1994. Responsibility for unreliable software. In *Proceedings of the conference on Ethics in the computer age* (*ECA '94*). Association for Computing Machinery, New York, NY, USA, 175–177. DOI:https://doi.org/10.1145/199544.199611