TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Seminar: Human Factors in Software Engineering

Maria Spichkova

TUM-I1216

Seminar: Human Factors in Software Engineering

Editor: Maria Spichkova
Institut für Informatik, TU München, Germany

October 1, 2012

Abstract

There are many definitions of human factors, however most of them (applying to the field of software and system engineering) are oriented on human-machine system operation in terms of usability of systems and programs, i.e. on those parts that are seen by (end-)user, but not by the requirements, specification and verification engineers. The fundamental goal of human factors engineering is to reduce errors, increase productivity and safety when the human interacts with a system. Engineering psychology applies psychological perspective to the problems of system design and focuses on the information-processing capacities of the human brain too.

This report presents the results of our first seminar on "Human Factors in Software Engineering" held in the summer term of 2012. The deliverables to be developed by the students were a learning module prepared by each student as final presentation and the documentation of the learning module in an essay (content of this report). The topics were:

- Topic 1: What does "Human Factor" mean?
  Student: Andreas Schmidt
- Topic 2: History of Engineering Psychology
  Student: Ruocong Shan
- Topic 3: Software Engineering: HA, HCI, and HF - Differences and Similarities
  Student: Susanne Brunner
- Topic 5: Human Factors Evaluation Methods
  Student: Rupert Dürre
- Topic 4: Human Error Paradigms
  Student: Duc Nguyen
- Topic 6: Human factors in safety-critical systems
  Student: Julian Sievers
- Topic 7: Specific features of UI for web-applications in comparison to other kinds of software applications
  Student: Tomas Ladek
- Topic 8: Usability in Software Engineering/ Development Methods
  Student: Fabian Wetekamp

Contents

1 Andreas Schmidt: What does "Human Factor" mean?

1.1 Definition of the Term "Human Factors"

The definition of the term "Human Factor" is not as easy as it may seem at first. In fact, there even is no universally accepted formal definition but many different approaches sometimes with a different focus. A study conducted at the Harry G. Armstrong Aerospace Medical Research Laboratory lists over 90 different definitions given in various literature connected to the subject [LP].

A very generic definition that focuses mainly on the biomedical and psychosocial aspects of human factors could be following [ea04, p. 1]:

Definition 1 *Human Factors are all physical, psychological and social characteristics of humans which have an influence on the interaction with and within a system or are being influenced by the system.*

A different approach, that sees Human Factors more as it's own field of research also including the practical application of the gained knowledge is given by Salvendy in his book "Handbook of Ergonomics" [(ed97, p. 4]:

Definition 2 *Human factors is the use of the knowledge about human abilities and limitations for the design of systems that are safe, efficient and comfortable for a human use*

One possible source of confusion in this context is the term "ergonomics", which can have a different meaning especially in european countries. Salvendy argues, that ergonomics and human factors are in fact different terms for the same field of study, a view that is also shared by the Human Factors and Ergonomics Society and the International Ergonomics Association [FS].

As one can see in the definitions, the term "Human Factors" covers a wide field of topics, in fact too many to be covered in this article. That is why the subjects in this paper are limited to certain fields which have the biggest influence from a software engineering point of view and are therefore of great interest to software developers.

1.2 Human Sensory System

1.2.1 Visual System

The Human visual system is one of the most important senses and from everyday experience it is very clear that a big portion of the information our senses gather about the surrounding world, comes from our visual system.

Biomechanics of the visual system

Electromagnetic energy in the form of light is transformed to neural signals in the eyes. This happens through photoreceptors which are located on the back inner-surface of the eye, the so called "retina". Situated on the retina are two different main types of photoreceptor cells, commonly know as "rods" and "cones". Rods have a higher sensitivity to light compared to cones. These lesser sensitive cone cells can also be further divided in three different types which have a unique sensibility for different wavelengths: "Blue"-Cones which peak at a wavelength of 440nm, "Green"-Cones (530nm) and "Red"-Cones (560nm) [Sch11, p. 77]. Since the receptors have different responses for a stimulus in a given wavelength, the brain has enough information to infer the wavelength of the signal thus allowing for the perception of "colour" rather than just plain luminance. This process is only possible with a minimum of two different kinds of receptors with a distinct receptability depending on the wavelength in order to allow an unambiguous interpretation of the information. This is one reason why colour perception does not work in bad lighting conditions where only the cones are sensitive enough to respond.

Influence on System Development

One thing that is most important to know about the human visual system in a human factors context is that it is by no means a device of accurate measurement. This fact can easily be demonstrated by a multitude of optical illusions of which a very impressive one is the "Adelson Koffka Ring" (cf. Figure 1.1).

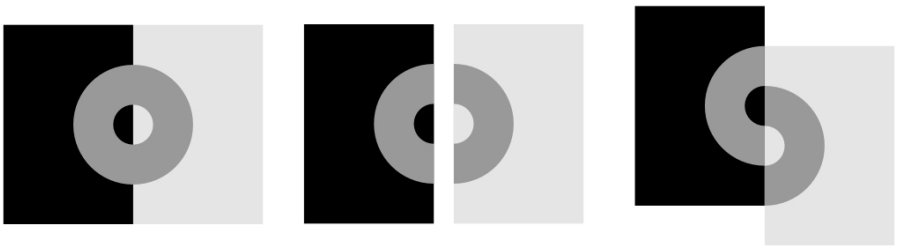

Figure 1.1: Adelson Koffka Illusion. The greyvalue of the ring appears to be changing when it is sliced in two halfs, even though it stays the same.

However not only the perception of greyscales, also the perception of colour can vary depending on the surrounding which is a differnt effect, known as "colour constancy" [Gol01, cf p. 119]: A red ball will always appear red to the spectator, irregardles of the lighting conditions, because it is red in relation to it's surrounding. In general the visual system is more suited to "compare" values rather than to absolutely measure them. Like all of our senses, vision only gives a functional working representation of our environment, rather than an absolute representation.

To deal with the special properties of the visual system in software engineering, there exist some different guidelines for the use of colours, to ensure a comfortable usage of the interface [Ben05, p. 126]

- Limited Use of colours. A maximum of 5±2 colours should not be exceeded.
- Avoidance of complementary colours (eg. blue/orange or red/green combinations)
- Consistency in Colour Coding and use of colour conventions which takes advantage of the fact that in most cultures, specific attributes are assigned to certain colours, like "hot temperature" or "danger" for red, a "safe state" for the colour green, or "caution" for yellow etc.
- Display Text with high contrast, preferably black on white to increase readability

1.2.2 Auditory System

Another sensory system that plays an important role in the interaction of humans with their surroundings is the auditory system.

Biomechanics of Hearing

Soundwaves are waves that travel through a medium like water or air which can be detected and processed by humans with their auditory system. The waves travel through the ear canal to the tympanic membrane which they set in motion. This movement is transferred by a mechanical system of small bones to the cochlea, a small spiral shaped chamber with tiny receptor hairs on it's inside walls. These hairs move according to the sound waves and produce a nerve signal that is transferred to the brain by the auditory nerve.

A very special fact about the auditory system, that distinguishes it from the other senses like the visual system is it's omnidirectional property which means, signal detection is not influenced by the orientation of the head relative to the source of the signal. However very much like the visual system, or in general all senses of the human body, the auditory system is everything but an accurate device of measurement. This can easily be demonstrated by two interesting facts about hearing:

1. The perception of loudness and the actual sound intensity are not correlated, meaning that the perceived loudness of a sound at 80dbA is not twice as loud as a sound of 40dbA
2. The perception of loudness is also influenced by it's frequency: of two sounds with the same sound pressure, the one with the higher frequency will be perceived as louder.

Some sounds can be inaudible in the presence of other sounds, when their frequency components are too similar, a process known as "masking". On the

other hand our auditory system has some kind of filtering system to distinguish signals from noise, in other words one is able to focus ones attention on certain parts of the auditory signal.

This mechanism of selective hearing can be seen in the so called "Cocktail-party effect" which probably everybody has already experienced. In a noisy surrounding like a party with a lot of sources of noise, it is fairly easy to concentrate on a specific conversation and filter the voice of the conversation partner from the others. Studies have shown, that this ability increases, when the "targeted" sound source and the masking sound source are spatially separated, thus giving the auditory system additional information in the form of interaural differences in the signals [Gol01, p. 540f]. Further factors that make it easier to separate sound sources are spectral and temporal separation and also the intensity of the sound. The louder a sound signal is compared to the co-occurring noise, the more likely is the sound perceived as a unique source of sound[Gol01, p. 552]. As a principle for design, Wickens suggests that a sound should be at least 15dB above the masking noise in order to be heared [Wic04, p. 96].

Applications in Software Engineering

Why is it important to know about the auditory system, when most of the interaction in current applications works purely visually? The answer is simple: The design of effective alarms. Auditory alarms or signals are in general superior in some ways over visual signals. The most important aspect is the fact that due to the omnidirectional property of our auditory system, alarms are sensed regardless if the user is oriented towards some userinterface like a computer screen. That means if there is an important alarm like a fire alarm, an acoustic signal should be the means of choice, preferably coupled with a redundant visual signal. Wickens cites in his books the works of Patterson, in which he discusses some criteria for good alarms [Wic04, p. 98]:

1. An Alarm should be audible amongst the background noise, thus avoiding the effect of masking
2. It must not be above the danger level of hearing
3. The sound should not be startling or abrupt
4. It should not interfere with other auditory signals by making them harder to understand or to interpret.
5. The alarm must be informative , meaning that a (trained) user can associate the alarm with a meaning or a cause for the alarm.

1.3 Cognition

After talking about the perception of stimuli from the world around us, we will have a closer look on how these signals are processed.

1.3.1 Bottom-Up vs. Top-Down-Processing

So far we only focused on one way of perception, the so called Bottom-Up Processing, in which the plain stimuli detected by the recepting organs lead to a perception of the outer world with little or no cognitive processes involved. A different process is the Top-Down Processing of information in which knowledge, expectancies and desires heavily influence the perception and the interpretation of the stimuli. The knowledge of these two different kinds of information processing processes is important in the context of attention.

1.3.2 Attention

Attention is the focus of "mental resources at or on a particular task or object" [Ben05, p. 107]. Missing attention can often be a source of human error and it is influenced by a variety of factors: salience, expectancy, value and effort [Wic04, p. 123]. *Salience* is a typical bottom-up factor, meaning that attention will be drawn to a signal when it's very distinct from it's surrounding for example a bright blinking warning light, or one red object in a bowl of green objects. *Expectancy* and *value* are top-down driven, meaning that attention will be directed at places, where information is expected and is of some value to the user. Selective attention is also heavily influenced by the amount of *effort* it takes to obtain the information. For example people will check a display less often, if they have to turn around to see it.

These aspects should be kept in mind during the design of interfaces, especially when displaying important information like warnings, to the user.

1.3.3 Visual Perception

Perception is the process of extracting meaning from an array of information (eg. audio or visual) [Wic04, p. 124]. The visual perception is of all mechanisms in perception the best understood process [Ben05, p. 111].

A particulary interesting and very important topic connected to interface design are the "Gestalt laws of perception" formulated by a group of psychologists in the beginning of the last century. [Ben05, p. 114] These laws describe when a number of objects appear to be in one single group. Some examples (cf. Figure 1.1):

- Law of Proximity: Objects close together appear to be in one group.
- Law of Similarity: Objects similar in shape or colour like the dark and pale circles in *b)* tend to be grouped together.
- Law of Symmetry: Objects that are symmetrical to an axis are perceived as one group. The literals in *c)* appear to be in three groups of two.

These laws are of tremendous importance in the design of user interfaces, in particular the positioning of display objects and input controls.

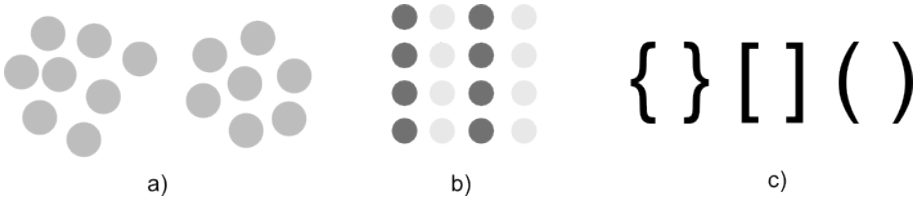

Figure 1.2: Some Gestalt laws of perception: a) Proximity, b) Similarity, c) Symmetry

1.3.4 Memory

Coming from a technical background the analogy of the memory layout of a modern computer and the memory organization of the human brain quickly comes to mind. Like the computer we also have a limited working memory we use for information processing and short term storage and a much bigger memory for long term storage, the long term memory.

Working Memory

A widely accepted theory about how the working memory functions was first suggested by Baddeley and Hitch in 1974. This theory splits the system in 4 components [Eys10, p. 211ff].

1. A "central executive" as controlling unit
2. A phonological loop in which the order of words is preserved, for example by a nonverbal rehearsal with ones "inner voice"
3. The visuo-spatial sketchpad for spatial and visual information, i.e. a route when walking
4. Episodic Buffer for complex scenes or episodes. The episodic buffer was added by Baddeley in 2000 and is a system that can integrate and briefly store information from various sources, like the phonological loop or the visuo-spatial sketchpad, in the form of short episodes [Eys10, cf. p 221]. This system enables us for example to remember a story or the scene of a movie.

Some aspects of high interest in Human factors research are the limitations and capabilities of the working memory. First let's have a look on the capacity. It is commonly believed that the average amount of information that can be stored in the working memory is 7 ± 2 chunks. A chunk is any meaningful entity: the random letters "D L U V" represent 4 chunks, whereas the word "door" consisting of 4 letters represents only one chunk. The grouping of symbols can

also have an influence: “5 8 7 3 2 9” has double the number of chunks compared to “58 73 29”. In general, familiarity and an association between the single units make up a chunk [Wic04, p. 130].

Information does not stay in working memory forever, it gradually gets lost without a continuous repetition or “maintenance rehearsal”, which is for example the repetitive reciting of items with ones inner voice. Studies suggest that the half-time for information in working memory is about 7 seconds for 3 chunks and 70 seconds for 1 chunk. That means information can even get lost, when the time between the information acquisition and the effective usage is too big [Wic04, p. 131]. This knowledge gives us some best practices in software engineering that should be followed. It is always desirable to minimize the load on the working memory and to use a design that makes use of the principles of chunking. An example could be the display of phone numbers in a database application in which the digits are grouped in pairs of two. Also, it is good practice to provide visual echoes for performed actions and reminders for sequential actions, since the user might get confused with the steps, while he is performing his task.

Long Term Memory

The long term memory is used for information storage and retrieval, the process of storing information is called learning. Forgetting is the fail of memory retrieval. A piece of information will be lost, when the specific memory is of weak strength, meaning it has a very small frequency of use, when it has too little associations to other memories, or when there are other interfering associations.

A piece of information is easier remembered when it is meaningful and concrete rather than abstract and relatively organized. The implications on software design in this case are the use of standards, for example the similarity of user interfaces in graphics programs which makes orientation in them easier, or the use of memory aids that trigger associations and help the user to remember steps they have to take in a program.

1.4 Human Error

A popular saying claims that the cause of most computer problems is about 40cm in front of the screen. Human error is an omnipresent problem that is responsible for many failures in complex systems. Two famous and very dramatic examples are the reactor incident at Three Mile Island in 1979 and the reactor catastrophe in Tschernobyl in 1986.

In general, errors can be divided in two different classes depending if they were a result of an intended or an unintended action. Errors because of intended actions are the result of a lack of knowledge. If the cause is an unintended action, it might either be confusion, for example because of badly arranged user interface elements or a failure in prospective memory, meaning a task was simply forgotten.

To cope better with human error, here are some rules for software design, similar to what Reason and Norman already suggested in 1990 [Ben05, p. 389]:

- Promote good conceptual models of the system, so the operators have a better understanding of the consequences of their action
- Simplify the structure of tasks
- Exploit the Power of constraints ("Trust no user input")
- Design for Error, plan for Error recovery
- Comply to accepted standards for example colour coding

1.5 Stress and Workload

A big source of human Errors is also a high level of stress. Stress can not only be psychological stress caused by arousal, or the cognitive appraisal of a task, but there are also environmental stressors like bad air quality, motions like vibrations, or thermal stress.
If the workload is too high, which means there are too many tasks to perform in an inadequate amount of time it also causes stress with extensive consequences: Poor decision making, a decrease in accuracy and high selectivity in input. Possible solutions for this problem are on the one hand a better task organization and on the other hand the use of automatiation to ease the workload on the user.

1.6 Conclusion

In this paper we have looked at different definitions of the term human factors and elaborated some of it's important aspects, like the influence of the different sensory systems on information retrieval and the cognitive processing of these informations. Also we talked about stress and workload and its influence of human failure.

The "Designing for Humans" factor of software engineering is a very important part that should not be omitted since human factors can have a high influence on the usage of software and the performance reached with it. Only the exact knowledge about the limitations, behaviour and other characteristics of humans allows the design of software that is safe, efficient and easy to use.

2 Ruocong Shan: History of Engineering Psychology

2.1 Introduction

Engineering psychology [Se68] is the science of human behaviour and capability, applied to the design and operation of systems and technology. As an applied field of psychology and an interdisciplinary part of ergonomics, it aims to improve the relationships between people and machines by reading equipment, interactions, or the environment in which they take place. The work of an engineering psychologist is often described as making the relationship more user-friendly.

Engineering psychology was created from within experimental psychology (see also [WH91]). It started during the World War II. However, long before the World War II, it has already taken place in matching equipment requirements with the capabilities of human operators by changing the design of the equipment. The real beginning can lead back to the end of 19th century. Therefore, the historical development of engineering psychology can be divided into 4 time sections:

- End of 19th Century
- World War I
- World War II
- Present

2.2 End of 19th Century

In a certain sense it is correct to say that people have been concerned with engineering psychology of a sort ever since man began fashioning implements for his own use. Nonetheless, engineering psychology has emerged as a separate discipline only within the past few decades. It was not until the end of the nineteenth century that the first systematic investigations were conducted on man's capacity to work as it is influenced by his job and his tools.

Frederick Winslow Taylor (March 20, 1856 - March 21, 1915), who was an American mechanical engineer and regarded as the father of scientific management, made the empirical studies of the best design of shovels and of the optimum weight of material of each shovel for handling different products, such as send, slag, rice coal, iron ore. Taylor's interests, however, were primarily in

rates of doing work and in the effects of incentives and worker motivation on rates of working. His study was sought to improve industrial efficiency.

About 10 years later, Frank Gilbreth (July 7, 1868 - June 14, 1924) began as a bricklayer, became a building contractor. He reduced all motions of the hand into 17 basic motions, such as grasp, transport loaded, and hold. He sought ways to make bricklaying faster and easier. Later in 1990, he set a firm foundation for this filed with his classic study of bricklaying. This pioneering work of Taylor and Gilbreth was the beginning of the branch of industrial engineering now known as time and motion study.

In the years that followed, time and motion engineers developed a number of principles of motion economy, of the arrangement of work, and of work design that have been widely applied throughout modern industry. Insofar as they have focused on human capacities and limitations and have used this information to redesign the machine, the task, or the work environment, it is correct to say that time and motion engineers are predecessors of the modern engineering psychologist.

Still, the primary emphasis in time and motion engineering has been on man as a worker, that is, as a source of mechanical power.

2.3 World War I

When the United States entered World War I in 1917, a group of psychologists under Robert M. Yerkes was organized as the Psychology Committee of the National Research Council. In volunteering their services to the military establishment, they were met at first with considerable scepticism about what they could do of any value in the hard business of war. Gradually the psychologists were able to make some substantial contributions and eventually win the enthusiastic endorsement of the military services.

By and large the psychologists in World War I were concerned with such things as the selection, classification, and the training of recruits, and with morale, military discipline, recreation, and problems of emotional stability in soldiers and sailors. A few of them, however, encountered problems of a different sort-those in which the design of machines and equipment had to be related to the user.

2.4 World War II

After the armistice in 1918 this pioneering work in engineering psychology was almost entirely abandoned. A few scattered studies appeared between the two world wars under the auspices of the Industrial Health Board and the Industrial Fatigue research Board of the Medical Research Council (Great Britain). At that time the machines and problems foreshadowed by World War I reappeared in profusion. Radar, sonar, high altitude and high speed aircraft, naval combat information centres, and air traffic control centres placed demands upon their human operators that were often far beyond the capabilities of human senses, brains, and muscles. Operators sometimes had to look for targets which were all

but invisible, understand speech against backgrounds of deafening noise, track targets simultaneously in the three dimensions of space with both hands, and absorb large amounts of information to reach life-and-death decisions within seconds. As a result, bombs and bullets often missed their mark, planes crashed, friendly ships were sunk, and whales were depth-charged.

Having entered the war before the United States, Great Britain faced these problems first and established a pattern that, in broad outlines, was followed later in the United States. The Medical Research Council was responsible for sponsoring much research on the Flying Personnel Research Committee, the Royal Naval Personnel Research Committee, and the Military Personnel Research Committee.

Although entering the conflict later, the United States met problems equally urgent and dealt with them in substantially the same way. The National Defence Research Committee through the Office of Scientific Research and Development set up numerous research contracts in universities and industries to study these problems. All three military services incorporated civilian and military scientist-psychologists into their research and development laboratories in order that research findings would be put to immediate use.

Before the birth of human factors, or ergonomics, in World War II, emphasis was placed on "designing the human to fit the machine." That is, the emphasis was on training. Experience in World War II, however, revealed a situation in which systems with well-trained operators weren't working. During the two world wars there appeared a new class of machines: machines that made demands upon the operator, not only his muscular power but his sensory, perceptual, judgemental and decision-making abilities. For example, the job of a sonar operator requires nearly no muscular effort, but it makes several demands on his attentiveness and his decision-making ability. Problems of this type could no longer be dealt with by common sense or by the time and motion engineer's principles of motion economy. That is why we say, engineering psychology was created from within experimental psychology, which started during World War II (1940).

Since then, the engineering psychology is well developed. Motivation for the development of human factors and engineering psychology as disciplines has arisen from three general sources:

- Practical needs
- Technological advancements
- Linguistic developments

All the problems and tasks appeared in the two world wars, for example, are the practical needs. Experimental psychologists were brought in to analyse the operator-machine interface, to diagnose what was wrong, and to recommend solutions. This represented the practical need underlying the origin of human factors engineering.

A second motivation has come from evolutionary trends in technology. With increased technological development in this century, systems have become in-

creasingly complex, with more and more interrelated elements, forcing the designer to consider technology and automation. This problem has led systems designers to consider the analysis of human performance in different kinds of tasks. At the same time, with increased technology, the physical parameters of all systems have grown geometrically. For example, consider the increases in the maximum velocity of vehicles, progressing from the ox-cart; in the temperature range of energy systems, from fires to nuclear reactors; and in the physical size of vehicles, from wagons to supertankers and wide-bodied aircraft such as the Boeing 747. Particularly with regard to speed, this increase forces psychologists to analyse quite closely the operator's temporal limits of processing information. To the ox-cart driver, a fraction of a second delay in responding to an environmental event will be of little consequence. To the pilot of a supersonic aircraft, however, a delay of the same magnitude may be critical in causing a collision.

Finally, an influence for the growth of human factors has come from the field of information theory and cybernetics that began to replace the stimulus-response language of behavioural psychology after World War II. Term such as feedback, channel capacity, and bandwidth began to enter the descriptive language of human behaviour. This new language enabled some aspects of human performance to be described in the same terms as the mechanical electronic, or information systems with which the operator was interacting, which helped integrate humans and machines in system design and analysis.

2.5 Present

Since its birth, the field of human factors has evolved from a discipline applied primarily to aviation and weapons systems, to one applied to a much broader range of products with which people interact, including such things as toys, telephones, medical devices and cars. Since World War II, the growth of engineering psychology has been very rapid. At the present time, engineering psychology is most fully developed and exploited as a speciality. People increasingly interact with computers, or with other devices like automatic teller machines or databases, through computer interfaces. Hence, the topic of human-computer interaction has evolved into a major focus of human factors.

2.6 Conclusion

Despite the evolution of products, the fundamental importance of designing for the strengths and limitations of the human user remains constant, because the fundamental characteristics of the human user have remained pretty much the same, as have the basic goals of human factors. It refers to that we should continually focus on the human factors, and develop different system and technology around it.

3 Susanne Brunner: Software Engineering: HA, HCI, and HF - Differences and Similarities

3.1 Introduction

According to D. Te'eni [TCZ07], human factors can be found everywhere in the software engineering process. It begins with requirements engineering and ends with customer testing. Human factors also surround the engineering process since it is humans who develop software. Over the years this has drawn many researchers, and their results have been collected in three main categories: human aspects (HA) of software engineering, human-computer interaction (HCI), and human factors (HF) in software engineering.

HA focuses mainly on the human concerns of those developing a software product, whereas HCI and HF are nearly exclusively concerned with ergonomics for the end user. However the distinction cannot be made this easily since there are hardly any definitions done by an international body of recognized authority. While the area of HA is quite clearly divided from the other two, the distinction between HCI and HF cannot be made this easily. The lines get blurred even more because, while some authors give a personal definition in their works, others omit it entirely.

This paper intends to give a general overview of what the three areas are, how they are most commonly defined, and how they might distinguish themselves from each other.

3.2 Human Aspects of Software Engineering

Human aspects (HA) of software engineering are the most easily specifiable category amongst the three.

J. Tomayko mentions in the introduction to his book on HA that he "attempts to highlight the world of software engineering from the perspective of the main actors involved [...]: the individual, the team, the customer, and the organization" [TH04]. In general, it can be summarized as the study about human factors regarding those on the development side of the engineering process.

HA covers subsections of many areas of study, for example workspace ergonomics, team dynamics, code ergonomics, and ethics.

Workspace ergonomics is concerned with providing an employee-friendly furniture set-up that will help prevent typical office illnesses like repetitive strain injury or back problems. Several ISO norms give hard regulations, for example

ISO 13406-2 which sets baselines for brightness, contrast, and light reflection of LCD screens for office work.

Knowledge in team dynamics has become essential because any sizable software can only be created by teams. Next to different personalities, different nationalities and cultures may be involved in the engineering process. As such cooperation, and knowing how to foster said cooperation, is of utmost value (see also [TH04]). Code ergonomics consists of program comprehension, code inspections, and refactoring amongst others. Its intention is to make code easily readable because it is highly likely that the code will be maintained or altered by others in the future. Additionally, it will help polish and refine the code.

During the software engineering process, there may also arise questions of ethics. In 1999, the IEEE-CS/ACM joint task force on software engineering ethics and professional practices saw it necessary to copyright the "Software Engineering Code of Ethics and Professional Practice", a recommendation that gives guidelines for how software engineers should conduct themselves. It has been formulated on the basis of other codes of ethics, the Code of Medical Ethics amongst others, because software has long become an intrinsic part of life with far-reaching consequences [TH04].

All those subtopics are part of HA of software engineering; with the exception of workspace ergonomics they have also become part of standard works on software engineering. In most cases, however, they have not been mentioned as human aspects or human factors. Ian Somerville, for example, lists (cf. [Som07]) them under the header of project management (ethics and team dynamics) and best practices programming (code ergonomics); other authors use similar categories.

In general it can be said that HA of software engineering are seldomly studied as a human factors discipline, because the most relevant areas have already become part of regular software engineering studies. Works like Tomayko's [TH04] are the exception.

3.3 Human-Computer Interaction

In comparison to human aspects of software engineering, human-computer interaction (HCI) is only concerned with human factors of the end user, not the developers. The ACM special interest group on computer-human interaction defines HCI as "the discipline concerned with design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them" (Te'eni [TCZ07]).

Historically, the term HCI has evolved from studies of the Ergonomics Research Society which was founded in 1949. It was initially named "man-machine interaction"; for gender-equality purposes and in recognition of the particular interest concerning computers it has been renamed to "human-computer interaction" [DFAB04].

This renaming though laid the groundwork for future difficulties in a clear definition of HCI: What is a computer?

Some authors like A. Dix in the first edition of his book on HCI [DFAB04]

completely omit stating what the 'computer' part consists of. Judging by the input-output-equipment mentioned, it might be inferred that Dix is only talking about personal computers. By the third edition though he refers to a computer as "any technology ranging from the general desktop computer to a large-scale computer system, a process control system or an embedded system" [DFAB04]. Dahm [Dah06] has a similar definition; Te'eni on the other hand explicitly states that he restricts himself to office software and business applications only [TCZ07].

Independent from any definition of 'computer' though, nearly all works on HCI model the influences on HCI with graphics similar to figure 3.1. Figure 3.1 is based on the one by Te'eni [TCZ07], with some simplifications for easier readability. Human factors, psychology, computer science, and engineering are the areas from which HCI draws the most information, with design a close second. There are also various other influences, languages and sociology and semiotics only a small selection, but they are of lesser importance.

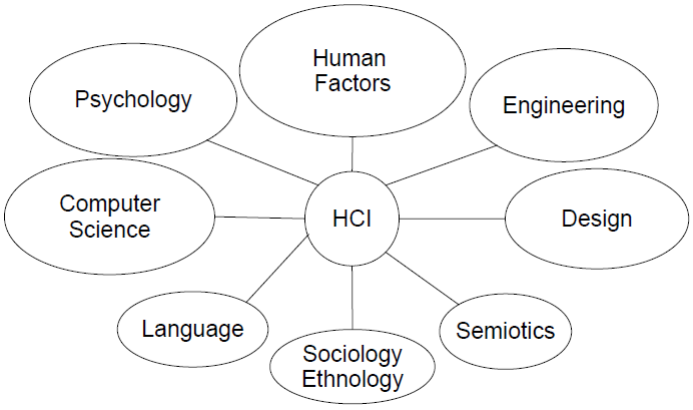

Figure 3.1: A selection of subjects influencing human-computer interaction

Overall, a general overview over the books by Te'eni [TCZ07], Dix [DFAB04], and Dahm [Dah06] has shown that the main topics, like usability paradigms, error paradigms, design paradigms, and evaluation methods, are independent of the scope of "computer", too. As such it stands to reason that the ACM definition given at the beginning of the section is adequate.

3.4 Human Factors in Software Engineering

In the literature research, no books on human factors (HF) in software engineering could be obtained. Works about HF in engineering were deemed irrelevant because they dealt with hardware engineering only. Several exemplars titled "Human Factors in Computing Systems" were volumes of CHI conference proceedings during the early nineties, and as such thought to be more relevant for HCI than for HF. Others were subtitled with variations of "human-computer

interaction", and thus could not be used for a definition of HF.

Due to the fact that HF is also called "ergonomics" in the European language area, works on software ergonomics were considered, too. Like Herczeg's "Software-Ergonomie - Grundlagen der Mensch-Computer-Kommunikation" [Her94] though, they contained the same problem of not being able to be assigned to one category only. None of them made a distinction between HF and HCI.

Judging by the level of overlapping in literature, it was assumed that HCI and HF are largely identical. As such, the definition of HCI should hold for HF, too.

In light of the problems with defining "computer" for HCI though, a definition independent of HCI, and more importantly "computer", should be found. The name "Human Factors in Software Engineering" suggests to use "software" instead.

Software, too, is subject to interpretation. For the most comprehensive definition possible it must be used in its most general sense, namely instructions telling a processor what operations to perform.

Figure 3.2 shows the different levels of software between user and hardware in a desktop computer. Since users only interface with the application layer, there exists the common misconception that only application software is software. In embedded systems, however, the application layer might be absent entirely, and thus such a narrowing of the definition of 'software' would narrow a definition of HF in software engineering too much.

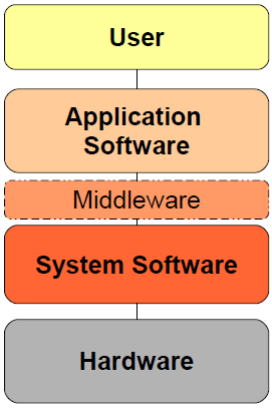

Figure 3.2: Layers of software between user and hardware in a desktop computer

As a result, HF in software engineering can be informally defined as "human interaction with machines which execute code". Upon closer inspection this definition is identical to the one given for HCI in the previous section since computing systems are nothing but "machines which execute code". However, it avoids the problems that arise upon defining what exactly a computer is.

3.5 Conclusion

Human aspects of software engineering, human-computer interaction, and human factors in software engineering are very closely related. HA deals with human factors concerning software developers, whereas HCI and HF are concerned with the end user.

In literature, HA are seldomly categorized as such; they are more likely to be incorporated in general works on software engineering under the header of project management and best practices programming. For HCI there exist many works; however the definition of "computer" may vary from office software only to personal computers only, to anything and everything including embedded systems. HF in software engineering has largely been used as a synonym for HCI in literature; however no works could be found which explicitly stated an identity between the two fields or a definition for HF in software engineering.

Since no literature definition could be found for HF and the HCI definition strongly depended on the definition of "computer", an informal definition based on "software" was postulated during the course of this work. However due to its identity with the chosen definition for HCI and the fact that HCI remained the same independent of the scope of "computer", it stands to reason that HF in software engineering and HCI are indeed identical.

4 Rupert Dürre: Human Factors Evaluation Methods

4.1 Introduction

Human factors look to enhance a system's performance, satisfaction and safety. Human factors evaluation methods are used to analyze a system regarding the human factors. They tell whether the design of a concept is adequate. This is done by comparing the objective performance data, which has been collected through tests, to the performance requirements, which have been determined before the construction of the system. Furthermore the evaluation methods show how well the decisions, which were made during the design process, turn out in the end. But human factors evaluation methods are not only used during the final testing and evaluation of a system. They are applied in redesign analysis and the technical design too. Therefore they are important in all design phases (cf. [Nem04]).

Thomas Edison, the great inventor, is an example for missed human factors: he invented not only the light bulb and the motion picture camera, he also invented the phonograph. His vision was a paperless office organized by letters recorded to cylinders and send to the recipient. But in fact the real desire of the customers was to listen to prerecorded music. Even after the desired use of the photograph was discovered Edison failed again: he thought, he could save money and maximize profit through recording lesser-known musicians. That way he probably saved some money, but lost a lot of profit, since the customers wanted to hear the well-known artists. Overall as you can see Edison missed to adjust his developed technology to customers' need and desires (see also [WLLGB04]).

When talking about the human factors in software engineering, many people just think about interface design. However, human factors go beyond the interface to design the task, that the system is wanted to do. Moreover, the definition of the course of human-machine-interaction is part of human factors, as well as the collection of data about the organization of people and technology. This means, that the methods want to evaluate the interaction of people with the system, the goals the users try to achieve and of course the relationship between both.

4.2 Fault Tree Analysis

As a practical introduction to human factors evaluation methods, an example we show (ex. Figure 1). The figure contains a Fault Tree Analysis (FTA). The root of the tree is a database failure screen and the FTA aims to identify

the basic events that cause the database error. Diagrams are often used to visuals and simplify the modeling and analysis of complex software systems. This diagram represents a technical analysis for the problem's occurrence (cf. [Cha96]).

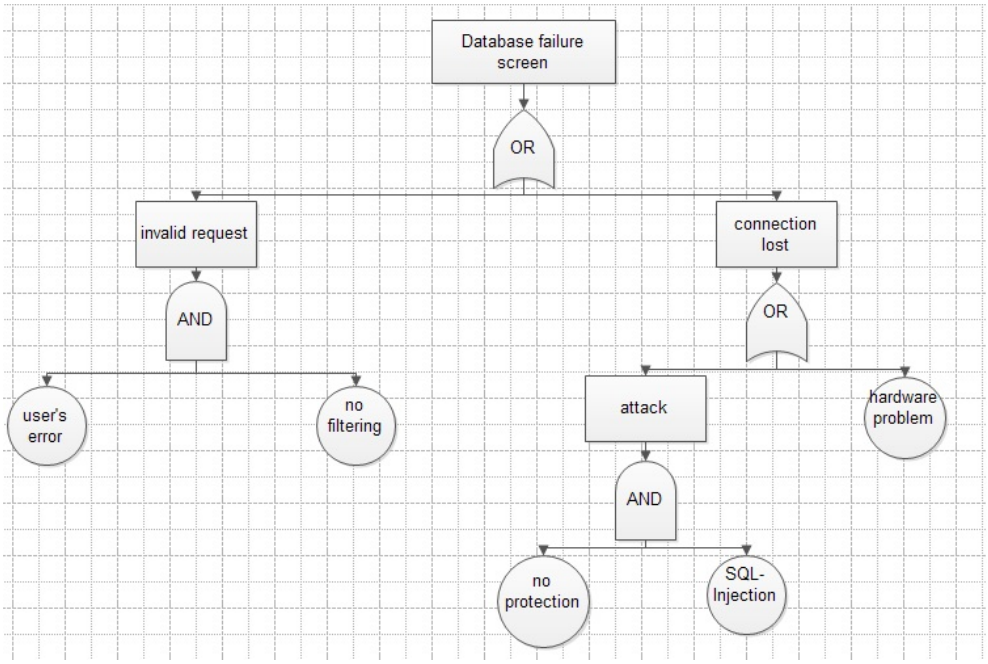

Figure 4.1: Fault Tree Analysis for the evaluation of occurrence of faults

Since Human Factors target to evaluate the human-system-interaction, it is obvious that the diagram from Figure 1 has to focus intensively on the user's error event and errors, that occur during this interaction. This event has to be considered accurately. Therefore this event is splitted into more sub events that contribute to the error occurring during the interaction between user and system. In this human factors failure evaluation, it might be useful to change the main event from the failure screen to a more general event that includes more possible errors in human-system-interactions. Therefore we changed the diagram from Figure 1 to fit the human factor analysis requirements. The result is presented in Figure 2. It is striking that the FTA is the same technique for engineering and human factor evaluation proposal. Thus it is a representative example for many human factors methods that were originally used during the engineering process or the engineering based evaluation of a system. Many engineering methods have been converted to human factor evaluation methods through focusing the analysis on human factors. Remarkable is that the human factor might play into that adoption of technical methods: people are used to work with these methods, so they take them out of their original context and use them to analyze different topics [Cha96, Nem04].

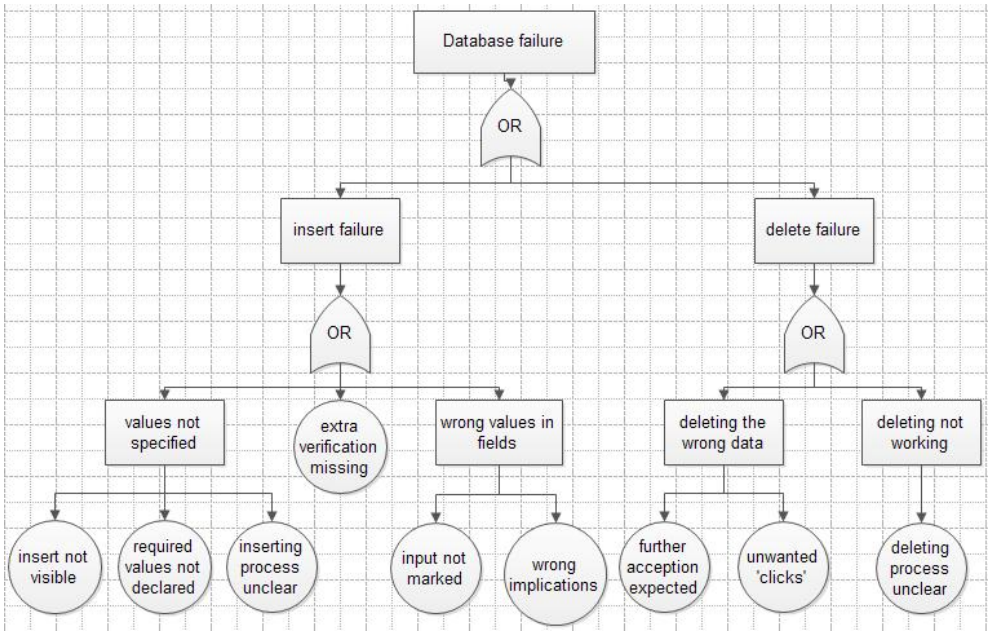

Figure 4.2: Fault Tree Analysis for the evaluation of occurrence of faults

As a graphical technique the FTA contains different elements. You have seen the most important elements in the both FTA diagrams in the beginning. Additionally they are listed in Figure 3. The usage of the elements will be defined in the following and illustrated with the example diagram in Figure 2 (see [Cha96] for more details).

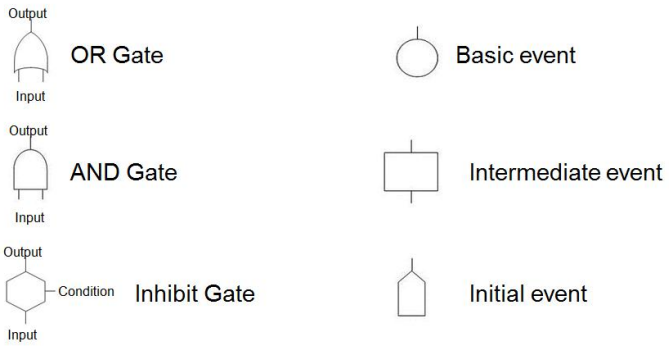

Figure 4.3: Fault Tree Analysis - Graphical elements

Generally a FTA consists out of four steps. At the beginning the main failure event has to be defined, since it will be the failure event we focus on. It can be an actual event that has appeared on the running system or it can be an imaginary event on a system that will be developed. Moreover, the input for the FTA can be the result of a risk analysis. The defined failure will be the

root of the fault tree later on.

The second step is to add events and conditions that could have caused the failure event. The nature of the causes is dependent on the focus of the analysis. In human factor evaluation the analysis will concentrate on the errors occurring during the human-system-interaction.

Afterwards we have to assign the appropriate logic gate to the defined cause events and condition. If both cause events have to occur to induce the considered failure event, we combine the events with an AND-gate. If either one can cause the failure event by itself we use an OR-gate to connect the events, causes and conditions.

Those two steps are repeated for each event that arises until we have exhausting information for the complete FTA. When this point is reached we simply put our information together and organize it in a tree: therefore the main failure event is used as the root and all defined causes and conditions will be put into the tree as they were defined during the previous procedure.

At the end the FTA provides a diagram that represents the collective faults, or impairments to performance, that can result in a dysfunction. Moreover it shows how a series of events can lead up to the considered failure. In the further process it is demanded to anticipate and reduce those potential sources for undesirable events (more details are presented in [SSB+04]).

4.3 Human factors evaluation methods - Approaches

After presenting the FTA from the point of view of a human factor evaluation method, more approaches to analyze human factors in (software) systems will be given. Before that some general information about human factors evaluation methods will be provided. Basically there are three kinds of evaluation methods. At first a theoretical analysis can be done: thoughts on a system that is currently or will prospectively be developed are the base of the analysis. This approach is often performed in early design and development stages to specify the design goals during the systems. Also it is used to clarify whether the currently defined design goals meet the requirements of the system.[WLLGB04]

Another way to evaluate systems is by using simulation. Simulation can be a computer simulation, a mock-up simulation or a combination of those two simulations. Computer-based simulation is the faster, more accurate and at times less costly approach. Mock-up simulation means building a physical prototype and it is preferable when a family of related systems has to be evaluated. This approach is often used during the development of a system. Through simulation it is easier to compare the current design of a system to the requirements that were defined at the beginning.

The last approach is to execute actual operations on a system. Therefore the system has to be running. This approach is pretty similar to mock-up simulation, except the system is more finalized and the evaluator has less control over the test conditions. The simulation as a human factors evaluation methods will be described in the end of task analysis [Nem04].

Often it is difficult to categorize a Human Factor Evaluation Method to one

of those approaches, because they appear interlaced. Moreover they may be used in every design step and be influenced by each other.

4.4 Task analysis

The most important method to evaluate a system is the task analysis. The purpose of the task analysis is to understand and represent human and system performance in a particular task or scenario. It can be described as "the study of what an operator [...] is required to do, in terms of actions and cognitive processes, to achieve system goals"[Cha96, p.36]. So, a task analysis describes the human-system-interaction to understand how to match the demands of the system to human capabilities. Therefore it is called "activity analysis"[WLLGB04] sometimes. Three terms often used in task analysis are goals, functions and tasks. Those terms will be clarified now to prevent misunderstanding. A goal is the reason or end condition for performing a task. Functions represent the general transformations a user has to perform to archive a goal and tasks are the specific activities a user has to do to carry out a function. So a goal does not depend on technology, it remains constant, but technology can change the tasks (see [WLLGB04]).

Generally the task analysis consists of four basic steps: define the purpose of the analysis, collect the required data, summarize the collected data and finally make further analysis with the data.

4.4.1 Defintion of the purpose of the task analysis

At first the purpose and the required data of the task analysis has to be defined. This is critical since a task analysis can last for a longer time and be very costly consequently. Therefore it is necessary to describe the task that should be analyzed.

Generally tasks can be divided into physical tasks and cognitive tasks. To visualize those two terms think of the swipe touch gesture on smartphones: the physical task is to set the speed, distance and direction of the swipe gesture to call a specific functionality(for more details cf. [WLLGB04]). The cognitive task would be the decision what the values of the speed, distance and direction should be. Hence a human factor specialist is demanded to pay strong attention to cognitive component if his task analysis concerns a complex decision making, problem solving or diagnosis or if a large amount of prior knowledge is needed to perform tasks. Additionally a large and complex rule structure based on specific situations requires increased compliance by the specialist.

Tasks can be described by several different ways. In the following four different types of describing task information will be defined:

The first described important information collected in many task analyses are hierarchical relationships, which describe the composition of tasks out of sub tasks and how tasks combine to functions (for more details see [WLLGB04]). In the smartphone example the "take a picture"-function is a combination out different tasks like: release key lock, open camera application, frame the picture and press trigger. Release key lock is defined through multiple sub tasks, since

you first have to press a button and then pull a slider over the touch screen. Through the description of the hierarchical task structure, the many occurring sub tasks become understandable. Another advantage is that the hierarchical relationships provide a powerful hint for designing training program, because natural groupings of tasks are identified.

Another approach to describe tasks is information flow, which describes the communication between people and the system. Moreover the roles that the people play within a system are considered. In software engineering the creation of system with multiple different users and privileges is a important subject. Therefore information flow is very useful in this area (cf. [WLLGB04]).

Describing the task sequence can also be used to obtain information about a task. Here the order and relationship of tasks over time is taking into consideration. An example is the "take a picture"-function from [WLLGB04]: the goal of taking a picture would not be archived if a failure in the task sequence would have been happened and the user tries to open the camera application before the key lock has been removed. This shows that a specific task order is needed to archive a goal. The information included in a task sequence is the goal of the task, the starting event of task sequence, the duration of a task, the concurrently performed tasks and a sequential relationship, which describes the dependencies between the tasks. The question behind a task sequence is whether the system is efficient and if people like to work with it.

The last type of information on a task is the location and environmental conditions, which describe the physical world in which the tasks occur. For the human factors evaluation considering software systems this is the least important task description technique.

All tasks are described from different perspectives through those four task descriptions. This is necessary for a comprehensive task analysis. After the purpose of the task analysis and the required data is identified, task data needs to be collected, summarized and analyzed.

4.4.2 Collection of data

The second step during task analysis is the collection of data. Data about a task in a system is collected through extensively interaction with multiple users. The approach to collect the data depends on the information required for the task analysis. At best a human factor specialist gathers information during observation and questioning of users as they perform a given task.

During observations users are monitored while using an existing version of the system. Therefore many different types of users have to be found and afterwards those users are asked to perform some specific activities under a variety of different scenarios. The observer follows the work, asks questions if needed and tries to identify different methods that are used for accomplishing a goal. Observations have many advantages, especially to interviews or focus groups: at first they can be performed either in the field where they usually are done or in a simulated or laboratory situation. Moreover sometimes things that people say they do, do not match with what they do. For example they omit crucial parts of their work. Further they might have difficulties imagining

(new) techniques. But since the structure of users work is often revealed in their thought, observations alone are not enough (see also [WLLGB04]).

Another data collecting method is the think-aloud verbal protocol. In this technique the user thinks out loud while performing various tasks and provides in this way insights into his underlying goals, strategies, decisions and other cognitive components. Verbal protocols can be categorized in three types: prospective, concurrent and retrospective. In prospective verbal protocols the users think aloud while imagine to execute a given hypothetical task. It might be difficult for the user to imagine the technique to use. Concurrent verbal protocols are obtained during the task performance and therefore they are difficult to obtain sometimes, because the user may have difficulties verbalizing their thoughts while performing complex tasks. During retrospective verbal protocols the user talks about how he performed a task. Basis of the user's review might be a video made during performing the task or his memory. Since this type of think-aloud-protocol is easier to the user, it has been showed (cf. [WLLGB04]) that the data received in retrospective protocols is the most useable, despite the errors occuring when reminding from long time memory.

Unstructured and structured interviews are other methods to collect data: the user is interviewed and asked to describe the general activities they perform using the system. It is necessary that the human factor specialist does not only ask about how the user performs a task, but also about the user's strategies and preference. By that the analyst should note point where the user feel uncomfortable, fails to achieve their goals and shows a lack of understanding. The difference between both interviews is that the unstructured is more like a free conservation, because there is no particular method in use for structuring the interview. The questions asked are more general like: "How do you feel ...?" and "What do you ..?". More details are presented in [WLLGB04].

Structured interviews [WLLGB04] use a predefined template of questions that the analyst will be asking. The questions and methods used to structure the interview make it more efficient, complete and comparable to other interviews. In this case the human factor specialist prepares questionaires and notes and then conducts several interviews with each user. The interview is recorded sometimes to have a better opportunity to analyze it afterwards. Often hierarchical relationships work well as description of tasks when using interviews to collect information, because interviews can easily be structured with questions about the relationship between functions, tasks and sub tasks.

Surveys and questionnaires [WLLGB04] are the written counterpart to interviews. Since there are no meetings between the analyst and the user, it is time-saving, but also not as reliable as interviews. Often surveys are often collected through free online-tools. Of course there are many more ways to collect task data, but the most important ones have been picked out and explained before. After the data collection has been successful, the next step is to summarize data, since the gathered information has to be documented and organized.

4.4.3 Summarization of the collected data

Basically there are three different ways to summarize the gained data. The first option is to organize it static lists, outlines and matrices. At the beginning the gained data is written into lists, but when the hierarchical outline gets relatively complete, the data is transformed to outlines and matrices, where related information for each sub task is specified. In a matrix every task typically has one row and the associated columns describe the task in terms of information input, required actions and duration for example.

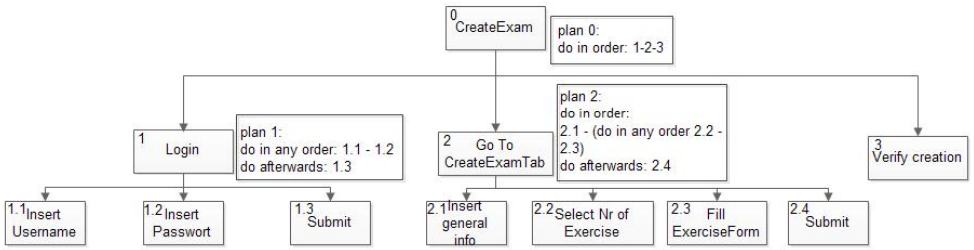

Figure 4.4: Hierarchical Task Analysis - Diagram

One further way to summarize data is to organize its hierarchies. This is a more useful way to describe tasks than lists and matrices, because tasks tend to have very complex hierarchical organization and this complex organization is easiest to describe and analyze when the information data is depicted graphically. That is why hierarchical chart and hierarchical networks are used often. In the following the Hierarchical Task Analysis (HTA) will be described as a representative of the hierarchical charts. The HTA organizes tasks as action to accomplish higher level goals. Therefore tasks are divided into sub tasks and plans. At first all task and sub task goals are determined and plans, which describe the task sequence and conditions for a task to be executed, are defined. A HTA helps with human factor evaluation, because from the HTA diagram it is able to analyze why the execution of a task is failing. So maybe the sequence of tasks described in the plans is too difficult for the user or the conditions do not fit with human expectations. An example for a HTA diagram is given in Figure 4. The task depicted is the createExam-Task, which is available in an imaginary online tool to create and evaluate exams (cf. [WLLGB04]).

The last task data summarization technique is to organize it in flow chart, timeline and maps. Those diagrams capture the chronological sequence of sub tasks as they normally are performed. Also the decision points for taking alternate pathways are depicted. A popular type of the flow chart is the operational sequence diagram (OSD). The OSD describes the interaction between the operators and the system and combines the events, information, decisions and actions involved in products use into a single diagram that is organized along a time line. As a sophisticated task summarize method, the OSA can be used

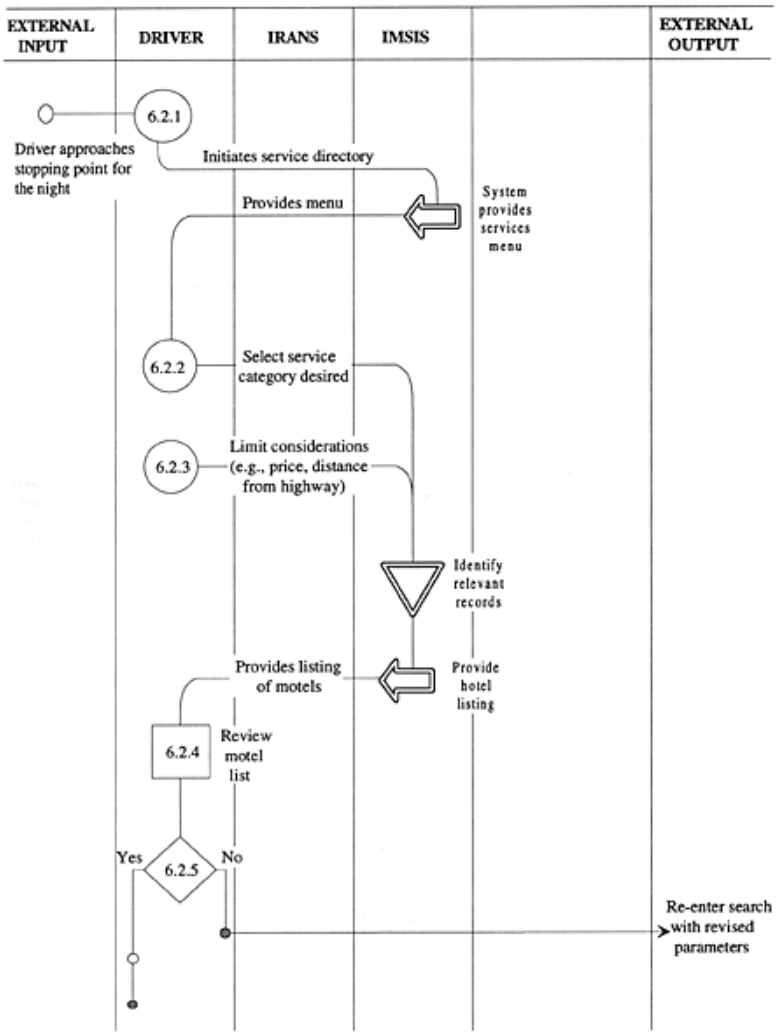

Figure 4.5: Operational sequence diagram from [oT12]

to develop a picture of a system operation that provides information on many different levels: it indicates the functional relationships among the multiple system elements, traces the flow of information, shows the sequential distribution of operations, identifies the input and output of subsystems and makes the result of alternative design configurations explicit. Figure 5 shows an OSD diagram. The background of this diagram is a mobile phone app, which helps on a trip to search for motels near to user's current locations. With the diagram the human factor specialist can identify discontinuities in timed relationship and look for opportunities to cut redundancies. Generally the time analysis shows the specialist whether the system is usable. For more details cf. [Cha96, Nem04].

4.4.4 Analyze the collected and summarized data

After the three ways to summarize data are explained, the human factor specialist can use the data collected and summarized before for further analysis. Besides the intuitive inspection of the gathered and organized data, that probably has been executed parenthetically in the previous steps also, there are formal methods to analyze task data. A few of them will be presented in the following.

The network analysis [WLLGB04] is a matrix manipulation and can be used to examine information flows in a network. Therefore the relationships between different identities in a graphic are transformed in a matrix. As example the matrix in Figure 6 displays the from-to-relationships from the flow diagram in Figure 7.

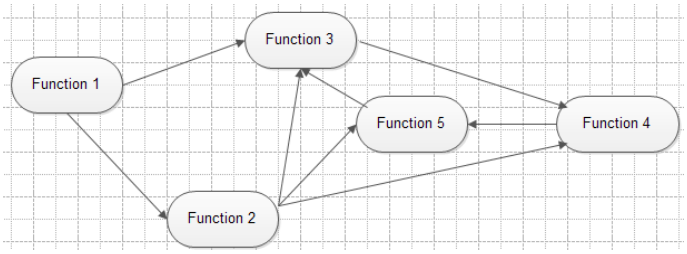

Figure 4.6: Network Analysis - Diagramm

	Function 1	Function 2	Function 3	Function 4	Function 5	Sum
Function 1		1	1			2
Function 2			1	1	1	3
Function 3				1		1
Function 4					1	1
Function 5			1			1
Sum	0	1	3	2	2	

Figure 4.7: Network Analysis - Table (from [WLLGB04])

From table it is easier to see that function 2 is the one providing the most output and function 3 is receiving the most input. The table is most useful when there a too many functions and the graph becomes more complex. A practical example for such an analysis is a website, where you have statistics on which page is visited how often and from where the visitors are directed from. If you have a page or function on the website this diagram will help you finding what problems occur and cause the lack of usage. The link to site could not be visible for the user or some browsers cannot display the link or the task sequence to get to the page/function could be too complex. Through this analysis, new approaches to solve the problem are imaginable.

Another approach in analyzing the collected task data is by creating scenarios out of the data (cf. [WLLGB04]). A scenario describes a situation and a specific set of tasks with an important role in the system. In software development scenarios are the first step in creating and evaluating the sequence of screens. Moreover scenarios define also tasks that users might be asked to complete in a Think Aloud Protocol or an Interview.

During the creation of scenarios only the tasks, which serve directly to users' goals, are retained. Scenarios can be categorized either in daily use scenario, that describe the common set of tasks that occur daily, and in necessary use scenarios, which describe infrequent used, but nevertheless critical sets of tasks.

Simulation and modeling [WLLGB04] can also be used as a method to analyze task data. Simulation generally predicts a dangerous or complex system or at least a part of it. It offers different possibilities: it can be used to either examine, evaluate or optimize a potential solution or compare alternatives. Some uncomplicated purpose can be evaluated with a simple "yes" or "no" to tell whether the concept does what it is supposed to do.

The mock-ups, models and prototypes (cf. [WLLGB04]) made by designers and engineers are used to simulate a concept. In that sense it is a design method. But each technique is also meant to collect and analyze data on human performance and preferences. Including the fact that increasing the efficiency of the human performance is the goal, it is obviously that simulation is also a human factors research and evaluation method.

The prototypes can be classified in two dimensions: the first dimension determines whether the prototype is physical or analytical. Physical means that the prototype approximates the product to evaluate its feel, demonstrate the concept and demonstrate how it functions. Analytical prototypes only represent traits for the system. The other dimension describes whether the prototype is comprehensive or focused. Comprehensive prototypes are a full-scale operation version of the product, for example a beta version of software. A prototype is focused if it implements only a few of the product attributes. Unit testing that is made during software engineering corresponds to focused prototypes. The focus often refers whether the function ("acts like prototype") or the composition ("looks like prototype") is presented in a prototype. In software engineering the "act like prototype" and the "looks like prototype" often are developed separate and then merge in the end. A concrete example is the creation of a complex JavaScript-based website. In some cases a designer gives a first design, a "looks like prototype", which is the basis for the further implementation. The designer and programmer both work separate on that basis: the designer polishes his design and the programmer implements the JavaScript behavior on the first design ("acts like prototype"). In the end they both merge their work and develop the finished system. For more details see [Nem04].

Another term often used with prototypes is the "paper prototype" [Nem04]. This is often used in development of software systems and it gives a preview of the user's interface.

After presenting the analysis of user's data you had an overview over the steps included in the task analysis, the most important method to evaluate system regarding human factors. The task analysis is very valuable, because

it contains and uses many other evaluation methods. Moreover it contributes to more evaluation techniques. For some it is a basically prerequisite. As shown in the FTA many evaluation methods originally were technical evaluation methods. Therefore it is necessary to really focus the evaluation on human-system interaction and the human capabilities while interacting in the system.

4.5 Conclusion

The presented human factor evaluation methods serve to satisfy and optimize the interaction between a user and a system. They help to understand failures in different parts of this interaction. By that, the methods give approaches to correct the failure and improve the interaction process. It is important to understand that evaluation does not mean that a system is evaluated after being finished, but it has to be steady progress from the first designing step on. It is preferable to include a human factor specialist early, because the earlier a change to a system is made, the less costly the change is. Moreover, it has a more personal reason: for human factor specialists it is easier to direct something in the right direction than to figure out bad human factor traits in a system in the end, because the engineer has put a lot of effort into the system and it is hard to disappoint him.

Thinking about the example of Thomas Edison in the beginning: a developer needs to satisfy the customer to build a successful system. Moreover the system has to be efficient usable. But in this point it is not only about the customer, also the developer desires to build a system that will be used and he can be proud of.

Further the human factor as a science needs evaluation methods. This fact is comparable to the psychology: at the beginning, psychology was not a real science. The intuitive interpretation of the human mind and behavior was not enough to gather general, objective explanations. By defining new research methods that had measurable results and describing the human being from different angles psychology rose to an accepted science.

5 Duc Nguyen: Human Error Paradigms

5.1 Introduction

For a long time, human has been accounted for a major proportion of system failures or disasters. In [Dhi04] there is a collection of statistic, which shows how much the share of human errors can be. In most cases, the human is responsible for 30% to 60% the total errors which directly or indirectly lead to the accidents. In some cases, like in aviation and traffic disasters, 80% to 90% of the errors were due to human. Therefore, if we can reduce the number of human errors, we can effectively lower the number of accidents and disasters as well.

Nowadays, a significant attention is being paid for studying the human errors and their causation. As a result, a number of error paradigms were introduced. They are the common patterns for the causation of human errors. Using these paradigms, we can quickly investigate the causes of a failure or accident. Moreover, since the errors in one paradigm often have some common solutions, it is also possible to quickly find a method to solve the existing problems or to prevent and avoid the probable errors.

5.2 Human error paradigms

Currently, four error paradigms were introduced [RR97]:

- The cognitive error paradigm, which focuses on the physical and psychological limitations of human.
- The engineering error paradigm, which focuses on the technical aspect of the system and interact between the human factor and the system.
- The individual error paradigm, which focuses on the unsafe acts of an individual and the reasons behind these.
- The organizational error paradigm, which focuses on the problems lie in the organization and management.

5.2.1 Cognitive error paradigm

The cognitive error paradigm is based on the idea of the mismatch between what people can do and what they need to do. This mismatch is the result of the physical and psychological limitations of human as mentioned in [RR97]:

- memory failure

- attentional failure
- information overload
- decision-making failure

The slips of action and memory lapses are the two most notable forms of cognitive errors.

For this error paradigm there is no direct solution since even a person with years of experience can still make this kind of mistakes unintentionally. However, we can improve the reliability of the human factor and reduce the possibility that such errors could occur. For example, we need to think about the tasks that need to be done, the context and situation where and when these tasks will be performed, and also about the people who will perform these specific tasks. The cognitive limitation of people has to be considered during this design phase of the system. Moreover, a system should be design in a defensive way, such that even when an operator forgets to make an action, or mistakenly tries to do something harmful, the system will not be badly damaged.

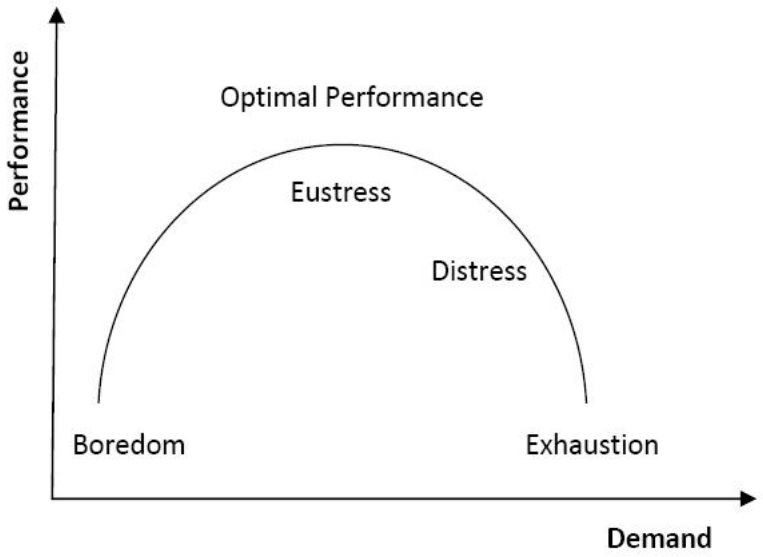

Figure 5.1: Relation between the effectiveness of human performance and the demand of work

In addition to this, it has also be proved that the likelihood of making these errors often get higher when people are in boredom or under stress [Rea88]. When the job demand is low, people tend to loosen themselves and therefore make attentional failure. When the job demand is too high, people often get stressed and thus, the chance of being overloaded with information and making wrong decision will be higher. If we can find the balance point of the work demand as shown in Figure 5.1, we can at the same time reduce the number of these errors.

5.2.2 Engineering error paradigm

Different from the cognitive error paradigm, which only looks at the human factor alone, the engineering error paradigm sees the human factor as one part of the system. The main idea of engineering error is that the human is typically unreliable and therefore the interaction between the human and the system makes the system flawed. As a result, the thought of "engineering out" human from the system via automation was used as a solution for this problem for a long time. In fact, the incorrect use of automation was one of the main sources for errors and failures. Nowadays, the weak "link" between the human factor and other components of the system is also considered as one of the main causes for engineering errors and failures. Because of this, the paradigm tries to improve the reliability of the whole system by strengthening the connection between the human factor and the system.

Errors related to automation

The use of automation in a system can introduce new kind of errors [Bra87]. The first problem with automation is that the automatic system itself still has to be designed and implemented by human. As a result, the automatic system is not completely reliable and flawless. In addition to this, the automatic system is normally more abstract and complicated than the original system and therefore more difficult to test and verify its correctness.

The next problem with automatic system is the change of human role in the system. Instead of directly interact with the system, people are now supported to supervise and monitor the automatic system, which can lead to two possible issues:

- Humans are generally poor at monitoring tasks since they are mostly repetitive and lacking of motivation. Beside of attentional failure, there is a very high chance that the operators overlook some important states of the system and result in making a wrong decision.
- As the operators are no longer interacting directly with the actual underlying system components, they can be "de-skilled". Even so, they are expected to act and keep the whole system running when the automatic system fails.

One possible solution for this problem is a periodical re-training for the operators, so that they will not only have a better and up to date understanding of the system but also be prepared for the case the automatic system stops working.

The last important problem with automation is its ability to influence the decision-making process of people. There were several experiments that prove the possibility of people making wrong decision because of the automatic system [BHM08, SMB99]. One of the reason is that most information of the complex system is hided by the automation "layer", only a subset of the actual states are available for the operators to see. If these states are displayed inaccurately or

incompletely, the operators can make the wrong decision. Therefore, to prevent this, the automatic system needs to be designed carefully and tested intensively.

Improving human reliability

The other main cause of engineering errors is the weak connection between the human factor and other components of the system. This often is caused by a bad human-computer interaction, or a bad system design that can easily confuse the users (or the operators) who are working with the system. Another example is a case when an operator needs to decide and act quickly but could not due to the fact that:

- the action which need to be done is too hard to complete in a short amount of time.
- the system displayed too much information that confuse the operator and the operator ends up taking too much time to make a decision.

To eliminate these errors we normally have to apply a number of methods to improve the usability of the user interface and make the system design better. These methods include usability tests, tasks and system analysis.

5.2.3 Individual error paradigm

The main idea of this paradigm is that accidents are caused by unsafe acts, such as violation of safety guideline or ignorance of warnings and hazards. Unlike errors in the cognitive paradigm, which are made without intention, errors in this paradigm are often the result of motivational and personality issues and they are caused on purpose. For example, because of work overload and dissatisfaction in working environment, a programmer could intentionally bypass some steps in his development process to quickly finish his work. This unsafe act of the programmer could result in buggy software or a more fatal software failure.

The individual error paradigm was traditionally used to blame a failure on to one person. The main concept was that errors are caused by a person not trying hard enough or not pay sufficient attention to the tasks [RR97]. With this, if a person made a mistake, he would be replaced with someone more suitable for the jobs or has to get through training again.

Nowadays, instead of blaming the person, the paradigm focuses more on understanding the reasons why people make mistakes or commit unsafe acts, and then tries to eliminate those reasons. The most applied solution is to build a better safety-culture for the whole company or organization, which encourages every person to follow the safety guideline and pay more attention to their work.

5.2.4 Organizational error paradigm

When investigate a failure case, the individual error paradigm is often viewed together with the organizational error paradigm. In compare to individual error

paradigm, the idea of the organizational error paradigm is that all the human errors need some kind of "conditions" in work context. These includes for example (as listed in [RR97]):

- poorly designed procedures
- unclear allocation of responsibilities
- lack of knowledge or training
- low morale
- poor equipment design
- time pressure

The organizational error paradigm believes all these conditions can be traced back to errors in the management and organization themselves.

Nowadays, there is a tendency to move away from blaming failures on individuals and toward the management and organizational issues. The errors in this paradigm are viewed as one of the root causes for errors in other paradigms.

5.3 Example case: Therac-25

The Therac-25 case (cf. [LT93])was one of the biggest and most disastrous of human error and engineering failure. Therac-25 is a radiation therapy machine, which is used in curing cancer. In 1985-1987 it had caused 6 accidents in which patients were overdosed with radiation. A typical single therapeutic doses consists about 200 rad (radiation measure unit) and anything above 500 rad can be considered deathly for the human body. In these accidents the patients received actually thousand times the normal dose, and most of them died because of cancer and pain.

Most of the error can be categorized as engineering errors. The most obvious mistake with the system is the bad design of the system. Therac-25 was an extension of the two previous models Therac-6 and Therac-20 but it still use the old software with only minor adjustments. It was the first machine that the company tried to mix two modes in one machine: one low energy mode and one high energy mode. In high energy mode, the filter plate must be placed correctly between the patients and the machine, so that the beam could be correctly used. In the accidents, because of some software failures, the high energy mode was used without the filter plate, this result in a deathly overdosed treatment. In the old models, the software sometimes causes errors, but it did not lead to overdosed accidents due to the hardware interlocks. In Therac-25, the company tried to replace these hardware interlocks with software checks. This design is lacking of defensive.

In addition to this, the Therac-25 made a very bad experience of human-computer interact. The first problem is its user interface, which sometimes displayed inaccurate system states. Moreover, the machine only showed error codes instead of full warning or error messages, and the codes were not even

documented. As the result, the operator of Therac-25 often found it normal to continue the treatment even after the machine showed warning messages.

The next errors can be found in the individual and organizational paradigm. The manufacturer of Therac-25 had over confident in their product. The company overlooked the reports and feedback from the accidents most of the time and therefore the actual fix for the machine was released very late. Together with very little of training, this caused the operators not aware of the importance of keeping the safety guideline and as a result, they violated many of the safety guideline. In some case, the operators conducted the treatment even when the video and audio monitoring, which were the only method to observe the patient in separated room, were not working.

5.4 Conclusion

As we had seen, the error paradigms are one of the most effective tools not only to investigate and understand the causes of an accident, but also to help improving the safety and reliability of a system. An important aspect of the paradigms that we need to have in mind is the relation and dependency between these paradigms. For example, an error in organizational paradigm can introduce another errors in cognitive and individual paradigm as already discussed. Moreover, one error can often be classified in more than one paradigm. Therefore, to use the human error paradigms effectively, we have to view an error under different paradigms and then apply all the possible solutions in order to fix and improve the existing system.

6 Julian Sievers: Human factors in safety-critical systems

6.1 Safety-critical systems

The intention of this chapter is to give a short overview about human factors in critical-safety systems. It describes the terms critical-safety system, safety and human error and goes into more detail about different safety theories. Both, reasons for system failure, and solutions are outlined, as well as human influence on system safety. Additionally, contributing factors to human failure are also addressed briefly. To understand the influence of humans on safety-critical systems, it is crucial to specify what a safety-critical system is, and then to analyse the role of the human in these systems. Per definition, accidents in safety-critical systems have severe impact on human life or the environment, including serious injuries, death and environmental harm. Nowadays, safety-critical systems increasingly depend on software functionality during operation, in the planning and the manufacturing process.

Safety critical software systems include some kind of the following functionalities, however this is not a complete list (for more info see [Lab12]). Critical software systems

- implements a critical decision making process,
- controls or monitors safety critical function,
- intervenes when an unsafe condition is present or imminent,
- handles safety critical data including displays of safety critical information.

Examples for critical safety systems can not only be found in large industrial facilities, but also in everyday objects. Medical devices like CADe (computer aided diagnosis) and CADx (computer aided detection), or the anti-lock braking system are just a few extract of the large variety. In high-hazard systems, these are systems with high potential risk but low actual risk [BSHL08], the nuclear and chemical industry, as well as aerospace have to be mentioned.

6.2 System safety

The term “system safety” describes the one hundred per cent probability that the whole system, not only its parts, work as intended [BSHL08]. This is the basis for every critical-safety system which is operated and build by humans, in

order to ensure failure free operation. Thus, safety is a state in which failure-free operation is possible and also meets additional conditions:

- correctness,
- robustness,
- freedom from failures.

Correctness means, that the system running as intended. This includes both, conformity with the system specification, and the fact that the system doesn't adopt any state other than described in the specification. So the system is doing what it should, but nothing more.

Robustness describes the ability to react to occurring failures and prevent a system breakdown by taking the correct actions to re-balance the system. Obviously these goals cannot be achieved and so we have to use a weakened expression of the mentioned conditions. This is described in DIN 40041, DIN is the German Institute for Standardization, which says that the above conditions only have to apply in a certain time interval.

Over the past year the role of the human in critical-safety systems has changed a lot and the human is more and more seen as a risk factor instead of a problem solver. This conclusion can be drawn because of the amount of "human errors" as cause of failure. In aerospace about 70 per cent and in nuclear energy about 52 per cent of the failures can be identified as human errors (see [BSHL08]). With the increasing complexity of today's software systems, the rate of human errors will become even greater over the next years if there is no trend reversal.

One possibility to handle this problem is to increase automation, which replaces the human and therefore eliminates human errors. The outcome of this is, that machines do the actual work and humans only monitor what they are doing. But automation of safety-critical processes can also expand the problems with human operators. Because of the rise of responsibility of human work, errors may lead to more significant accidents in the case of failure. This paradox is known as "irony of automation" [BSHL08].

6.3 Safety research

Although disasters do not happen daily, history proves that even in ultra-safe systems accidents only happen occasionally. The Bhopal disaster, Three-Mile-Island incident or the Chernobyl disaster are just a few examples. Due to these disasters it is crucial to investigate and find explanations to prevent avoid further accidents. Turner, Perrow and Weick provide three basic theories what effects can lead to disasters.

6.3.1 Man-made disasters

Investigation reports of accidents often point out problems of information flow and misinterpretation of failure supporting events although, at first glance, the catastrophe was unpredictable, a "fundamental surprise". Turner wrote in his

theory of "man-made disasters" [BSHL08] about this paradox and identified some reasons for all of the analysed disasters. Especially contributing factors which do not immediately cause errors, but have a large spatial and temporal distance to the trigger will be a problem for correct information interpretation. Reason goes more into detail about this fact and considers the background of an error as much as the error itself. He distinguishes between "active errors" and "latent conditions" [Rea90].

Active errors occur at the human-machine interface and are generally powerful enough to immediately cause unwanted consequences. Due to their easy identification, active errors are more in the focus of public discussion and commonly lead to penalisation of the responsible person.

Latent conditions do not cause accidents itself, but are a great contributing factor which reside unknown in the system until, in combination with an active error, lead to an accident. These contributing factors often have no foreseeable connections to the error, so safety is no primary goal of them. A latent condition can be found on every level of an organisation, examples include structural design or employee training and are for that reason hard to identify and prevent. Despite, or perhaps due to their distance from the actual accident, it is crucial to investigate, because a person who commits an active error always suffers from already existing latent conditions.

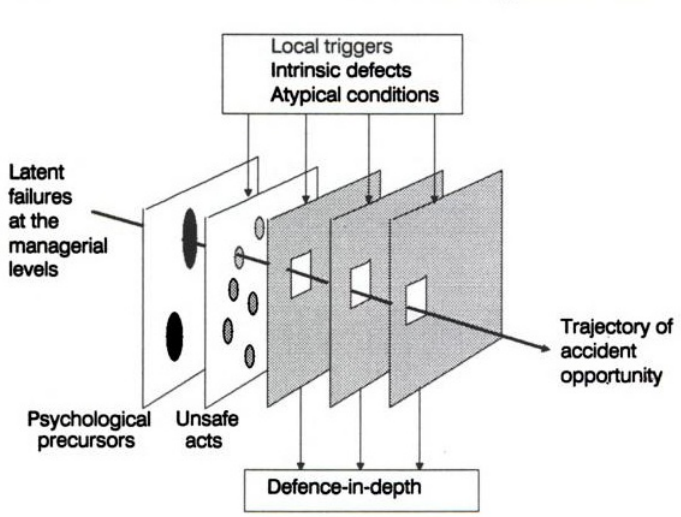

Figure 6.1: Swiss cheese model by Reason [Rea90]

Figure 8.3 shows how latent conditions and active errors contribute and then cause an accident. The slices in the model represent safety barriers in a system which prevent accidents. Without a security gap, an error committed on any level of the company cannot break through the the barriers and therefore not lead to a disaster. However, every system has at any time plenty of security gaps, as illustrated in 8.3. These are unsafe acts which did not directly lead to an error, but rest until some conditions come together. Additionally, only one small hole in each barrier is not sufficient, they have to be arranged one after another so that an active error can pass through all of them and thus result in a catastrophe.

6.3.2 Normal accident theory

According to Perrow and his "Normal accident theory" [BSHL08], system failure is a normal process and has to happen in every complex system. Perrow claims that unpredictable interdependencies result in errors and are natural consequences for every system with "complex interactions" and "closely tied components".

For humans, complex interactions are not obvious and their outcome cannot reasonably foreseen at that time. A heat exchanger for instance, which works as a heating unit, too. A malfunction of one component also affects another, at first glance completely different, part of the system. This problem is very hard to take stock of for the operator. If there are too many of these complex interactions in a system, then human errors will be more likely.

On the other hand, Perrow also identified closely tied components as possible risk factors for system failure. An example for closely tied components is the just-in-time production in the automobile industry. Many external suppliers deliver small parts of the system, which are the put together to form the whole car. Therefore, it is possible that delivery problems of one supplier can lead to a stop of the whole production line, because the cars cannot be completed.

This example of closely tied components in a system shows that there are benefits of this strategy (no storehouse costs) but malfunctions will have large impacts on the system. On the contrary, system design can also be loosely tied. Occurring events then have less impact on the system, because it is able to absorb errors until a particular point and re-balance the system. Buffer or duplicate systems are an example which can help to maintain system status. The system designer has to consider all these factors and find a balance of the pros and cons of closely or loosely tied components to ensure safety in all components.

6.3.3 High reliability theory

Weick's and Roberts's high reliability theory deals with organisations where fewer accidents then statistically predicted happen (see [BSHL08]). Their main interest focused on the failure-free operation in high hazard organisations, more precisely their research is based on observations on air craft carriers, the nuclear industry and the federal aviation administration.

Heedfulness was identified as main safety feature which depends on several characteristics:

- fault tolerance,
- no simple interpretations,
- flexibility in organisational processes,
- respect for technical knowledge.

Additionally, it is crucial for system safety that employees try to constantly reinterpret the system status to be able to rapidly react to malfunctions and therefore prevent accidents. The combination of these desired paradigms results in the high reliability of the examined area.

6.3.4 Accident prevention

There are plenty of ways to ensure system safety and prevent accidents. The whole scope goes beyond the overview of this paper, which therefore only describes some safety principles and possible safety management strategies to a limited extent.

One safety principle is for example, to design "fail safe" systems [BSHL08]. This involves predetermined breaking points and duplicate system components. The first named requirement targets the ability to prevent accidents despite of a malfunction, so it can balance the occurring defect and still work as intended. The second requirement builds upon redundancy. Systems of safety critical importance have to be implemented duplicate with the same or slightly different functions (but with the same outcome), so that a one component failure does not affect other connected components. Autopilots in aerospace are a suitable example for duplicate systems. Only if two out of three independent autopilot systems of an aircraft do work, the pilots are allowed to land the aircraft with the automated system.

Another safety principle attacks after an accident has happened. The main idea of "Defence in depth" [BSHL08] concentrates on safety features which become important on after another. A failure of the first safety barrier will not directly lead to an accident, but rather trigger the next barrier.

Other strategies to ensure system safety are described by Rasmussen [BSHL08]:
Feed forward control addresses detailed predictive strategies such as security and evacuation plans which are identified by risk analysis.
Feedback is generated out of operating experience event history analysis, so the knowledge of past is used to prevent accidents in the future.
In combination, the effective implementation of both methods contributes to more system safety.

6.4 Human error and disasters

As already mentioned the role of the human as risk factor is very problematic in critical-safety systems and accidents are just a matter of time. With regard

to judicial investigations it is crucial to mention, that not every human error is intended and therefore the perpetrator cannot be blamed for his actions. This means, there has to be a distinction between intended actions, they are called mistakes, and slips, which describes an unintended action. A mistake would for example be, if a the operator mixes up two levers, due to a lack of concentration, whereas a slip would be if he open a valve, because he does not know that it should not be opened at that time.

For error prevention it is important to work out the reasons for human errors. There are many different classifications about what causes human errors, one example is from Senders and Moray [BSHL08]:

Employees	Job	Equipment and Tools
Age	Arousal, fatigue	Controls, displays
Ability	Physical workload	Electrical hazards
Experience	Mental workload	Mechanical hazards
Drugs, alcohol	Work-rest cycles	Thermal hazards
Gender	Shifts, shift rotation	Pressure hazards
Stress	Pacing	Toxic substance hazards
Alertness, fatigue	Ergonomic hazards	Explosive hazards
Motivation	Procedures	Other component failures
Accident proneness		
Surrounding Environment		
Physical Environment	Social/Psychological Environment	
Illumination	Management practices	
Noise	Social norms	
Vibration	Morale	
Temperature	Training	
Humidity	Incentives	
Airborne pollutants		
Fire hazards		
Radiation hazards		
Falls		

Table 6.1: Causal and contributing factors for accidents by Wickens [Wic04]

“Endogenous errors” are based on physiological and biological factors, individual knowledge and skill, as well as information processing and employee motivation.

On the other side are the “exogenous errors”. They include organisational factors and the working environment, plus team based causes.

Some of the factors from each category are mutually supportive, such as a non-ergonomic working environment with physiological and biological factors. The complexity of a work task (exogenous), for instance, influences the motivation of an employee (endogenous) and increases error probability.

Table 6.1 gives an overview about contributing factors to human errors and represents the large field of human factor research.

6.5 Conclusion

This paper gives an overview on critical-safety system system and presents several explanation attempts for human errors, their causes and prevention methods. Safety research is a vital research field with increasing importance due to the human desire for safety, effects for the environment and economic losses. Furthermore, the increasing risk potential of critical safety systems requires more and more attention to ensure system safety. Thus, involving human factor methods and analysis gets more and more important nowadays, and is fundamental to meet the requirements of complex systems and to eliminate human errors as main cause for accidents. Additionally, human factors have to be involved even more, to meet the requirements of complex systems and eliminate human errors as main cause for accidents.

7 Tomas Ladek: Specific features of UI for web-applications in comparison to other kinds of software applications

7.1 Introduction

Web-applications assume a very special role in the software landscape. In it's early years the World Wide Web was thought of as a completely new medium and a definitive alternative to all software that until then existed mostly on individual computers only. In other words web-applications were designed so they could replace installed applications. One key aspect of this kind of software however, the UI, would not change much, because the human factors that come into play there also haven't changed, naturally. In the following will be summarized, what UI features web-applications make use of and a parallel between installed applications will be drawn.

All facts that will be presented are the summary of findings from these three books: *Grundlagen der Mensch-Computer-Interaktion* by Markus Dahm, *Usability – Nutzerfreundliches Web-Design* by Vittoria von Gizycki and Markus Beier and *Usability praktisch umsetzen* by Petra Vogt and Sven Heinsen.

7.2 Overview of web-applications

This section's purpose is on the one hand to define what the terms "installed applications" and "web-applications" in this paper stand for and what they are and on the other hand to give some explanation for the development toward the latter.

In this paper only "installed application" will be considered an alternative to web-applications and all comparison will be made only between those two types of software. Installed application is the term for a single program that needs to be installed prior to usage and which runs in it's own separate process (i.e. not as part of a process of the operating system) but on one machine only. Good examples are the well known Microsoft Office components like MS Word, MS Excel, MS Outlook, Adobe products like Acrobat Reader, Photoshop or Premiere Pro, instant messaging clients like ICQ, Skype, development tools such as eclipse or MS Visual Studio and even web browsers themselves like Firefox, Opera and Chrome. Games however, while technically also being installed applications, are a special case in terms of UI and aren't taken into consideration in this paper.

The term "web-application" in this paper is used for applications that strictly divide business logic and presentation in the following way: the business logic,

meaning the program's (complex) algorithms, runs on relatively powerful servers generally in one place in the world, whereas the presentation (or "user interface" - UI) is handled exclusively by relatively simple and weak client machines in a web browser, potentially located all over the world. The UI of web-applications is a website which is accessible by entering a URL[1] in a web browser. Focusing on the UI, the term website will in this paper be used as a synonym for web-application. Good examples of websites this paper is about are google.com, amazon.com, facebook.com, youtube.com, ign.com/boards, guardian.co.uk and similar.

The separation of the software aspects described in the last paragraph led to many advantages of web-applications over installed applications. To list a few: no need to install or update other software besides the web browser[2], fast access to very different services like social networks, information websites, forums, eCommerce sites etc., up-to-date information thanks to a dynamic connection between the user and the site provider and the possibility of personalising the viewing experience through browser settings. Being so versatile and facilitating the whole process of contact establishing and purchasing/selling, web-applications became also economically very valuable. All these aspects, in particular the user's rising need for up-to-date information, have caused the software landscape to increasingly favor web-applications over their installed counterparts.

7.3 Usability of websites

This section describes important UI features of websites and explains the human factors that play a role in web design and development in order to make web-applications usable.

7.3.1 Definition of usability

In it's definition of "usability" ISO[3] mentions three aspects under which a product (in this case a website) can be rated: effectiveness, efficiency and satisfaction. When a website performs well in every one of these aspects, it is considered "usable" or having the property of "good usability". Effectiveness of websites in case of information sites simply means that all information a users is looking for on the website is actually present. On eCommerce site effectiveness would be that a desired product can not only be chosen but also bought, etc. Efficiency measures the amount of employed resources (mostly time) by which a user can achieve the task he came to do on the website. Satisfaction cannot be easily measured and can only be statistically approximated as it depends on every user's individual pleasure.

[1] Uniform resource locator - "address" of a website on the internet, in the simplest case
[2] browser plug-ins may be installed additionally to view some web pages
[3] International Organization for Standardization

7.3.2 Structure of and navigation on websites

Installed applications and websites differ very strongly in the way users operate them. Installed applications often rely on a variety of menus and on buttons that change the program's mode and/or confront the user with different property windows and dialogue boxes. Websites work instead with fewer and smaller menus, on several pages or parts of pages rather than in modes and almost completely omit the use of property windows (or new windows in general). This is because when working with the web people associate new windows opened by the browser mostly with pop-up adds and many also install browser plug-ins to prevent their potential opening.

Structure

Designing a website with multiple webpages poses a challenge since all of the pages need to compose some sort of a reasonable structure. Failing to establish a clear site structure will lead to confusion of the user, which in turn decreases the efficiency of the website and therefore makes it less usable. Human short-term memory is known to be able to hold 7 ± 2 items, so while it's theoretically possible to navigate on this number of pages that are linked together randomly, a website with a page amount higher than that should help the user orient himself by ordering and linking the pages in such fashion, that he does not have to remember any one single page and still can navigate to any other page easily only by understanding the underlying structure. For the sake of simplicity or manageability websites tend to consist of rather small pages, but many in number. The usage of installed application, however, revolves mostly around one main work area that is modified by button presses or settings in property windows. Therefore the emphasis on structure lies more with web-applications and is critical for their usability. Examples of possible site structures are:

- Linear - simple sequential ordering of pages.
- Hierarchical - pages are ordered in a tree form.
- Network - hierarchical structure with links between some or all branches. This is the most common structure form since it makes navigation simpler and faster

Navigation

A linear structure is the most understandable and straightforward for humans - web browsers support navigating linearly forward and backward one page at a time with a forward and back button respectively. Most websites are making other navigation possible through menu bars and similar elements that are usually found at the top and left of the pages (this became a quasi-standard). Some standard navigation elements like a "contact", "help" or "shopping cart" link are expected to be found on every website (the latter in case of a eCommerce website). A site map can be found on some websites and is a full representation of the website's structure but it can grow too large and confusing and therefore

is not used very often. There are two major navigation styles that are chosen by web designers and developers:

- Horizontal bar at the top
  - advantage: saves space
  - disadvantage: only a very limited number of items next to each other fits on the screen
- Vertically ordered items on the left
  - advantage: very good readability
  - disadvantage: consumes potentially a lot of space of the page

The reason why left-aligned items underneath one another are readable well is that after completely reading an item the human eye can quickly jump one position downward and begin reading the next one, with the hight and the gap between lines as well as distance from the left border usually not altering from item to item.

Another fairly popular navigation element (in the hierarchical/network structure) is the "breadcrumb". It's a display of hierarchy levels that the user descended and optionally the level he is currently located on. The level descriptions are mostly clickable links, allowing for a quick way up the hierarchy, possibly skipping levels. The breadcrumb items are best placed horizontally from left to right, separated by a greater than sign (">") which has proven to be most understandable by symbolising the expected "motion forward" when navigating through web pages.

The last notable UI feature of websites' navigation is a search. It is being offered by an increasing amount of all kinds of websites and is considered one of the key navigation elements that can greatly influence the efficiency aspect of a web-application. By searching, the user can quickly find exactly what he is looking for and never have to worry about the website's structure and finding his way through. Very efficient search is the one implemented with the user in mind, meaning that it tolerates type errors, suggests similar search keywords and works with the user's terms (as opposed to company/industry terms). It is mostly placed somewhere on the website's entry point[4] and labeled accordingly. Installed applications do sometimes offer search of the program's functions as well, though it is not that common.

7.3.3 UI elements of (web-)applications

Due to the fact that web-applications developed out of installed applications they both have many UI elements in common, sometimes only slightly changed in their appearance. This mainly helps users recognise specific controls on websites like buttons they can press, checkboxes they can mark and drop-down-menus they can open and thus making the use of websites easier and more

[4] also known as root or home page

efficient. There are however differences in the usage (or usage frequency) of some UI elements that will be described in the following.

Use of multimedia and colouring

Primarily the type of used multimedia differs from installed application to web-applications. Web-applications tend to use more images, because humans are able to scan and understand them faster than text. This also applies to animations and embedded video (basically moving images). Colours are also used slightly differently - almost every website has it's own distinct "colour code", i.e. not more than two to three colours that are used consistently on every page. Compared to installed applications these colours are generally richer in contrast. Since one goal of all websites is to catch a visitor's attention and make him stay longer, it is advisable to focus on the visual perception (which is the most prominent for humans) and increase the website's attractivity in that way. Installed applications on the other hand utilise sounds, e.g. when warning a user, which is uncommon on websites.

Links

Links are a fundamental feature of the web and they should be and most often are used as frequently as possible. They minimise the effort with which a user can navigate on a website and directly increasing it's usability. For example while reading hypertext[5] users can quickly follow links that are attached to the words they directly read over. The majority of menu items are simple links as well. Links are mostly set apart from non-links by colour or other style properties so they are easy to spot. Most browsers also change the cursor when hovering the mouse over a link to indicate a clickable area. Installed applications can sometimes also feature links (e.g. in help sections, where the use of hypertext is reasonable) but they are far more seldom and their importance for usability is minimal.

7.4 Conclusion

After having taken a look at the features that make UI of web-applications different from installed applications, the question whether there are specific ones that are exclusive to one or another kind of software can be answered. The short answer is no - there is no feature of a website that cannot be found in some installed application. This can be accredited to the fact that web-applications originated from installed applications. Some UI concepts have just proven to be good and since the users remained the same, the human factors remained the same and so there is no need to change the proven UI concepts either, in the contrary - rather reuse them!

However websites do get used in a different way than installed programs. One reason is the different workflow that results from not having one main work area

[5] text containing links to other readable text

with different modes and a lot of property windows, but instead a lot of pages linked together in a unique structure and the need to navigate between them somehow (e.g. by the multitude of links). And lastly, the visual appeal of the two platforms is quite distinct as well (due to very differently dimensioned use of multimedia and colours), having a large effect on the usage experience.

8 Fabian Wetekamp: Usability in Software Engineering/ Development Methods

8.1 Introduction

Usability in software engineering is one of the most significant aspects of human computer interaction. It describes the process of the definition, measurement and improvement of the quality of a software's operability.

Over the last few years, Information Technology has taken an immense leap. The software market developed from a prosumer to a much bigger consumer market. The complexity of hard- and software has increased as well, technology that was a few years ago the standard for desktop computers (e.g. multi-core computing) is nowadays the usual configuration of a smartphone. Correlating with this development, the importance of usability for modern software products increased simultaneously. The term "Usability" is applicable to basically everything that is human-made. According to Rubin, "Usability is the ease of use and learnability of a human-made object" [RC08]. In the context of software engineering, usability stands for the quality of the human-computer interaction. But why did the factor usability became so important for software development?

For the major part of the modern civilization, software is a product for everyday usage. Developers face the challenge of integrating usability into software engineering and creating a satisfying user experience more then ever. So for good software, the usability is as important as the usefulness of the program. Besides, usability is increasingly seen as "marketable" (cf. [MR92]). The best-selling software products are the ones that are being considered having the highest usability. The usability of their products is for companies developing software often the key success factor when competing in a market (e.g. Apple, Dropbox, Google Docs). So when major software companies promote their product, they don't just show what the software does; they show how it's being done. For software developers, there are a lot of possibilities how to achieve high usability for their software, and how to integrate the usability engineering into the development process. The most important and validated patterns were collected, summarized and generalized by the International Organization for Standardization (ISO) in Geneva as the the international standard for human-computer interaction, named ISO 9241.

8.2 ISO 9241 - Ergonomics of human-system interaction

In 2006, the standard has been renamed from "Ergonomic requirements for office work with visual display terminals" to "Ergonomics of Human-System In-

teraction". The standard describes the requirements for usability in hardware, software and working environment and gives the usability engineer design recommendations for his project. The following figure shows the structure of the ISO 9241(cf. [**?**]):

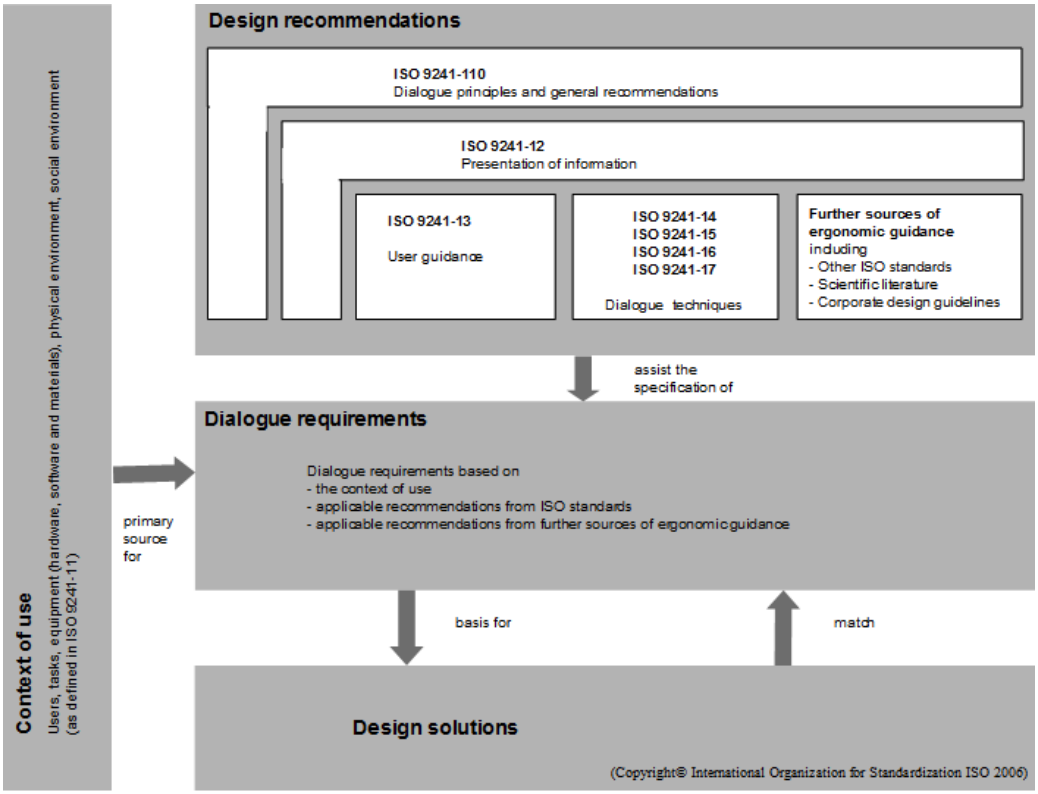

Figure 8.1: ISO 9241-1:1997, Ergonomics of human-system interaction, Part 1: General Introduction

The ISO 9241 states that there are three essential requirements software to be considered usable (cf. [92498]):

Effectiveness: The degree to which an interface facilitates a user in accomplishing the task for which it was intended.

Efficiency: The rate or speed at which an interface enables a user to accurately and successfully complete a task.

Satisfaction: A common reference to the set of subjective responses a person has when using a system.

Based on these requirements, the ISO 9241 describes the patterns for usability engineering. There are two essential parts in the ISO 9241 for usability engineering, the first one is the part 210. It provides guidance on human- system

interaction throughout the life cycle of interactive systems. The cycle described starts with understanding and specifying the context of use and defining the user requirements. Afterwards, the designer should produce design solutions to meet the user's requirements and in the end, evaluate the design solution against the requirements imposed before. If necessary, these steps are being iterated until a sufficient design solution is found (cf. [92410]).

The second part is the section 110 of the ISO 9241, which presents a set of usability heuristics that applies to the interaction of people and information systems. The standard refers to this interaction as a dialogue and describes seven dialogue principles:

- Suitability for the task (the dialogue should be suitable for the user's task and skill level)
- Self-descriptiveness (the dialogue should make it clear what the user should do next)
- Controllability (the user should be abled to control the pace and sequence of the interaction)
- Conformity with user expectations
- Error tolerance
- Suitability for individualization (the dialogue should be customizable to suit the user)
- Suitability for learning (the dialogue should support learning)

In conclusion, the part 210 shows the developer how to use the design solutions presented in the ISO 9241, and part 110 shows him the basic principles how good usability is being achieved. These two sections help the developer to become an idea how the integration of usability into development process can be realized.

8.3 Integrating Usability into software development

Curtis and Hefley specified, that the process of integrating usability into a software project is based on three key success factors [CH94]. First of all, the process for the user interface, the design for instance, has to be defined. Afterwards, this process has to be integrated with the remaining developing processes of the product. The third requirement states that during the integration of the usability, the project management should avoid making commitments, so an established project management discipline is necessary. It is very important to integrate usability early in the development process, because the further the development advances, the more costly it is to implement changes in the user interface. Consequently, usability has to be integrated in software development, quality assurance and project management, as Figure 2 shows.

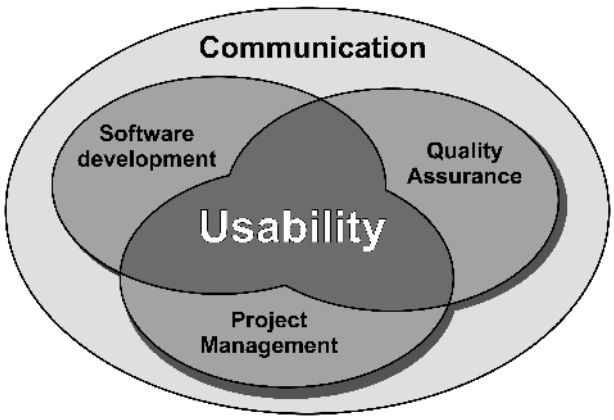

Figure 8.2: Usability Integration

Usability engineering itself is separated in three stages, the predesign stage, the design stage and the post design stage (see also [Nie93]). In the predesign stage, the usability engineer has the primary responsibility to gather analyze the user requirements and express these requirements in the product vision and use cases. The engineer sets usability goals by accumulating and analyzing data about the end user and the competitors. In the design stage, the engineer choses from various methods for designing the user interface, like parallel design, participatory design and iterative design. The final design stage is used for collecting data in field tests with the (almost) complete software. This data can be used for final adjustments or for future software projects.

After the integration of the usability engineering has been defined, the development of the user interface can begin, therefore the next section describes patterns for user interface development.

8.4 Methodologies and Processes for User Interface Development

The following three design processes are a small excerpt of a huge variety of development processes that are applicable for user interface (UI) development, but they are representative and show the relation of the UI Development methods to the ISO 9241.

The task-centered design process is structured around specific tasks the user wants to accomplish with the system. This design process is focused mainly on the user. The usability engineer creates representative tasks which describe who is going to use the system and what they intend to do with it. To use the task-centered design process, the tasks have to be chosen in the early stage of the development to have the right impact on the development. In the advanced development process, the task-centered structure helps to make essential design decisions and to evaluate the design.

RAD (Rapid Application Development) is an outgrowth of prototyping meth-

ods and conceptual work. Because software products are often deployed quickly, faster software development has changed from a competitive advantage to a competitive essential. This is the key success factor of RADical software development [BH94]. The development is based on evolutionary model that focuses on delivering versions of the product throughout the development (see also [BH94]). Each application is both initiated as a project and planned in a evolutionary life cycle, as Figure 3 illustrates.

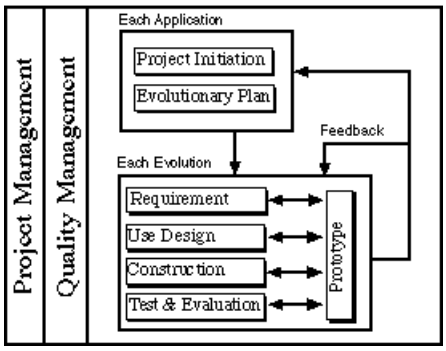

Figure 8.3: The Evolutionary Life Cycle

RADical software development uses continuous application engineering techniques and is performed by a dedicated professional team. The fast development is being achieved with time-boxed project management and powerful development tools. The benefits compared to other development methods are quality solutions in profound productivity results.

Essential modeling has a different approach to the point of view of the design problem. Instead of modeling how problem seems to be, the model describes what the system is intended for. As specified by Brown [Bro08], essential modeling involves three independent models.

The user role model covers the characteristics of the various users of the system. These characteristics can be for instance the degree of how technical experienced the expected average user for the system is. The essential use case model is a simplified and generalized form of use case. A typical use case for an email program for example would be to check the mailbox for new emails. The use context model is an abstract model of the architecture of a proposed user interface. Constantine refers to the use context model as “low-fidelity” prototype, because it doesn’t look much like a real screen layout or dialogue box design, but it already has essential elements needed to support the essential use cases (cf. [Con95]). These three processes represent valid development life cycles that integrate the usability engineering well into software development. The following section deals with the most common methods that are being used in usability engineering itself.

8.5 Formal Methods for Usability Engineering

The main goal of formal methods for usability engineering is to create design specifications that are known to be unambiguous and can be possibly proved to be correct. The methods are divided into inspection methods (without end users) and test methods (with end users).

The heuristic evaluation is the most common inspection method. Usability specialists evaluate the user interface step by step and judge whether each dialogue element follows established usability principles. Afterwards, the evaluators communicate with each other and compare their results (see also [NM90]). This method can be used very early in the development process and helps to identify problems in the user interface early and effectively. The biggest disadvantage of the Heuristic Evaluation is that it does not include the end user, so the user's needs are basically being ignored.

The cognitive walkthrough method is a task-oriented inspection method with which the analyst explores the system functionalities. It simulates a step-by-step user behavior for a given task and focuses on the cognitive theory by analyzing the mental processes required of the users. This is being achieved by providing an obvious way to undo actions and offering limited alternatives. A cognitive walkthrough is independent from end users and a fully functioning prototype helps designers to take on a potential user's perspective (cf. [LW97]).

The third inspection method is the action analysis, which focusses is more on what the practitioners do than on what they say they do. In the context of usability engineering, actions are tasks with no problem-solving and no internal control structure, they are mainly intuitive interactions between the user and the system (e.g. clicking on a button with a mouse). The task is being broken into individual actions while calculating the times needed to perform the action [CMN80]. The action sequences that a user performs to complete a task are closely being inspected by usability engineers and evaluated in terms of efficiency and effectiveness. This gives the inspector a deep insight into the user's behavior, but the method is very time-consuming and the evaluator needs a high expertise.

Thinking aloud is the most valuable usability engineering test method. It involves having an end user continuously thinking out loud every step he takes while using the system. The most valuable results are achieved when the users are directly saying what they are doing, because the working memory contents matter and retrospective reports are much less useful (see also [BB85]). This method reveals why the users do something and gives a direct feedback to the evaluator and a very close approximation to the individual usage. The results have to be perceived carefully, because the fact that the users are being forced to concentrate and may feel observed can have a negative influence on the validity of the test method.

Field observation is the most simple test method. It involves visiting users in their workplaces while usability experts are documenting the interactions of the users with system without attracting attention to avoid interfering with the user's work. Noise and disturbance can also lead to false results, so the observer should be virtually invisible to ensure normal working conditions. Data logging

and video recording are valid alternatives to the direct observation. This test method is only applicable in the final testing and to get valid results, a relatively high number of users is needed (cf. [Row94]).

With questionnaires [Lew95], the opinions of the users about the user interface are collected and evaluated. This is the best way to measure subjective satisfaction of the users and possible anxieties. The results of the questionnaires are useful for studying how end users use the system and to compile statistics. Because this is an indirect method and the responses in a questionnaire are always subjective, this method has a relatively low validity and does help to identify only a low number of problems relative to the other methods.

8.6 Conclusion

Usability is now recognized as an important software quality attribute, and over the last few years, this development reached the end users. So for most software-centric products, the key buy-decision is being made based on the usability of the product. For usability engineering in software development, it is important to set usability goals early in the development stage and based on these goals, the development of the user interface has to be integrated in the development process right from the beginning to achieve a satisfying user experience. In the decision-making for the best fitting development method for usability engineering, design recommendations of the ISO 9241 should be taken in account to analyze and to evaluate the considered development methods.

For formal methods in usability engineering, the most important conclusion is that there is no definite remedy and no all-in-one solution, because every formal method has its advantages and disadvantages. The worst danger is the belief that a single technique will provide sufficient results for usability engineering, ergo a combination of convenient formal methods is reasonable.

Bibliography

[92498] ISO 9241-11. Ergonomics of human-system interaction, part 11: Guidance on usability, 1998.

[92410] ISO 9241-210. Ergonomics of human-system interaction, part 210: Human-centered design for interactive systems, 2010.

[BB85] C. Bereiter and M. Bird. Use of thinking aloud in identification and teaching of reading comprehension strategies. *Cognition and Instruction*, 2(2), 1985.

[Ben05] D. Benyon. *Designing Interactive Systems - People, Activities, Contexts, Technologies*. Pearson Education Ltd., Essex, 2005.

[BH94] S. Bayer and J. Highsmith. RADical Software Development. *American Programmer*, 7, 1994.

[BHM08] J. E. Bahner, A.-D. Hüper, and D. Manzey. Misuse of automated decision aids: Complacency, automation bias and the impact of training experience. *International Journal of Human-Computer Studies*, 66(9):688 – 699, 2008.

[Bra87] L. Brainbridge. Ironies of automation. In Rasmussen J., Duncan K., and Leplat J., editors, *New Technology and Human Error*. John Wiley and Sons, 1987.

[Bro08] D. Brown. The how to of essential modeling, 2008.

[BSHL08] P. Badke-Schaub, G. Hofinger, and K. Lauche. *Human Factors: Psychologie Sicheren Handelns in Risikobranchen*. SPRINGER Publishing Company, 2008.

[CH94] B. Curtis and B. Hefley. A wimp no more: the maturing of user interface engineering. *interactions*, 1(1):22–34, 1994.

[Cha96] A. Chapanis. *Human Factors in Systems Engineering*. Wiley-Interscience Publication, 1996.

[CMN80] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, 1980.

[Con95] L.L. Constantine. Essential modeling: use cases for user interfaces. *interactions*, 2(2):34–46, 1995.

[Dah06] M. Dahm. *Grundlagen der Mensch-Computer-Interaktion.* Pearson Studium, München, 2006.

[DFAB04] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human Computer Interaction.* Prentice Hall, 3rd edition, 2004.

[Dhi04] B. Dhillon. *Engineering Usability: Fundamentals, Applications, Human Factors, and Human Error.* American Scientific Publishers, 2004.

[ea04] P. Badke-Schaub et al. *Human Factors - Psychologie sicheren Handelns in Risikobranchen, 2. Aulage.* Springer Verlag GmbH, Berlin, Heidelberg, 2004.

[(ed97] G. Salvendy (editor). *Handbook of human factors and ergonomics (2nd edn.).* John Wiley & Sons INC, New York, 1997.

[Eys10] M. W. Eysenck. *Cognitive Psychology - A Students Handbook, 6th Edition.* Psychology Press, 27 Church Road, Hove, East Sussex, 2010.

[FS] Human Factors and Ergonomics Society. About hfes - what is human factors/ergonomics? , https://www.hfes.org//web/abouthfes/about.html.

[Gol01] E. B. Goldstein. *Blackwell handbook of Perception.* Blackwell Publishers Inc., Oxford, 2001.

[Her94] M. Herczeg. *Software-Ergonomie.* Addison-Wesley, 1994.

[Lab12] Critical System Labs. What is critical safety software http://www.criticalsystemslabs.com/pgs/What.html, July 2012.

[Lew95] J. R. Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.*, 7(1):57–78, 1995.

[LP] D. Licht and D. J. Polzella. *Human Factors, Ergonomics, and Human Factors Engineering: An Analysis of Definitions.* Crew System Ergonomics Information Analysis Center (CSERIAC), Harry G. Armstrong Aerospace Medical Research Laboratory.

[LT93] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *Computer*, 26:18–41, 1993.

[LW97] C. Lewis and C. Wharton. Cognitive Walkthroughs. In *Helander, M., Landaeur, T.K., Prabhu, P. (eds.) Handbook of Human-Computer Interaction. Second Edition*, pages 717–732. Elsevier, Amsterdam, 1997.

[MR92] B. A. Myers and M. B. Rosson. *Survey On User Interface Programming.* ACM Press, 1992.

[Nem04] C. P. Nemeth. *Human Factors Methods for Design: Making Systems Human-Centered*. CRC Press, 2004.

[Nie93] J. Nielsen. *Usability engineering*. AP Professional, 1993.

[NM90] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, CHI '90, pages 249–256. ACM, 1990.

[oT12] U.S. Department of Transportation. Development of human factors guidelines for advanced traveler information systems and commercial vehicle operations: Task analysis of atis/cvo functions http://www.fhwa.dot.gov/publications/research/safety/95176/appendd2.cfm, July 2012.

[RC08] J. Rubin and D. Chisnell. *Handbook of usability testing : how to plan, design, and conduct effective tests*. Wiley Publ., 2008.

[Rea88] J. Reason. Stress and cognitive failure. In Fisher S. and Reason J., editors, *Handbook of Life Stress, Cognition and Health*. John Wiley and Sons, 1988.

[Rea90] J.T. Reason. *Human Error*. Cambridge University Press, 1990.

[Row94] D. E. Rowley. Usability testing in the field: bringing the laboratory to the user. In *Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence*, CHI '94, pages 252–257. ACM, 1994.

[RR97] F. Redmill and J. Rajan. *Human factors in safety-critical systems*. Butterworth-Heinemann, 1997.

[Sch11] T. Schubert. *Wahrnehmung und Aufmerksamkeit*. Springer Verlag, Berlin, Heidelberg, 2011.

[Se68] D.L. Sills and R. K. Merton (editors). *International Encyclopedia of the Social Sciences. Engineering Psychology*. 1968.

[SMB99] L. J. Skitka, K. L. Mosier, and M. Burdick. Does automation bias decision-making? *International Journal of Human-Computer Studies*, 51(5):991 – 1006, 1999.

[Som07] I. Sommerville. *Software Engineering*. Pearson Education, 8th edition, 2007.

[SSB+04] P. Salmon, N. Stanton, C. Baber, G. Walker, and D.Green. Human factors design and evaluation methods review. Technical report, Defence Technology Centre, February 2004.

[TCZ07] D. Te'eni, J. Carey, and P. Zhang. *Computer Human Interaction*. John Wiley & Sons, Hoboken, 2007.

[TH04] J. E. Tomayko and O. Hazzan. *Human Aspects of Software Engineering*. Charles River Media, Hingham, 2004.

[WH91] C. D. Wickens and J. G. Hollands. *Engineering Psychology and Human Performance*. Prentice Hall, 3 edition, Janury 1991.

[Wic04] C. D. Wickens. *An Introduction to Human Factors Engineering - Second Edition*. Pearson Education Inc., Upper Saddle River, New Jersey, 2004.

[WLLGB04] C. D. Wickens, J. D. Lee, Y. Liu, and S. Gordon-Becker. *An Introduction to Human Factors Engineering*. Pearson Prentice Hall, 2004.