Extended System Engineering – ESE: Integrating Usability Engineering In System Engineering

Avigdor Zonnenshain[1], *Avi Harel*[2]

[1] *Rafael, P.O.Box 2250, Haifa 31021, Israel*
[2] *Ergolight Ltd., 6 Givon Str., Haifa 34335, Israel*

Abstract
The goal of SE is to enable the realization of successful systems. Common SE practices define how people should use the system, but they disregard the performance limitations of the human operator and they do not bother to make sure that people actually use it according to the specification. Typically, system engineers ask usability professionals to make recommendations on usability issues, but usability professionals are typically concerned about ease of use during the initial stage more than about user errors. Users waste time trying to recover from their errors, resulting in productivity loss, reduced satisfaction and accidents. Although the bottleneck of the interaction is the user, engineers typically use to measure the system performance and reliability with the user outside the system boundaries. This article presents a methodology for extended system engineering (ESE). It presents the principles and means for integrating usability in system engineering, so that user errors are eliminated, and their negative effects are reduced.

The Problem of usability assurance

At exactly 4:00 a.m. EST on March 28, 1979, the main feedback pumps in the secondary cooling system of the Three Miles Island nuclear plant failed. Due to a design mistake, a pressure valve did not close after being open, and the reactor became overheated. Due to design mistakes, important indicators were missing. The scope of the accident became clear over the course of five days, as a number of agencies at the local, state and federal levels tried to diagnose the problem and decide whether the on-going accident required a full emergency evacuation of the local community, if not the entire area to the west/southwest. (http://en.wikipedia.org/wiki/Three_Mile_Island_accident).

This is one of many examples of a system failure that developed to an accident, because the operators, overwhelmed with irrelevant information, could not find the information relevant for the problem identification. Many other examples may be found in http://www.ergolight-sw.com/CHI/Company/Articles/ESE-Incose2008-P192.pdf .

The need for a methodology for usability assurance

It is typical to system developers to blame the users for their mistakes, even though they were misled by improper system design. This approach is convenient for the developers, but by blaming the users we disregard the reasons for the mistakes, and consequently we avoid preventing the next mistakes.

Managing the risks of usability deficiencies

The method proposed here for usability assurance is based on the common methodology of risk management. Risk management is a structured approach to managing uncertainty related to a threat, a sequence of human activities including: risk assessment, strategies development to manage it, and mitigation of risk using managerial resources. (http://en.wikipedia.org/wiki/Risk_management#Risk_retention). The previous section demonstrates the risk assessment of usability defects. The remaining of this article discusses strategies for managing these risks, and the implied requirements for managerial resources.

Limitations of common QA practices

The formal definition of quality is about the utility in using a product or a system. For example, this is the definition in http://www.chesapeakebay.net/info/qa_glossary.cfm. ISO/IEC 9126-1classifies software quality in a structured set of six characteristics, one of them is usability. The problem is that common QA practices do not target the usability requirements. Too many products fail due to usability limitations after being qualified by formal QA procedures: because they are useless, because they are too complex to use, because the enable critical user errors. Too many systems intended to protect security installations work perfectly at the QA qualification stage, but fail when they are needed, because psychological aspects were not considered at the design phase. Too much time we waste trying to find out why the software behaves so strangely, or what should we do in order that the TV will show a picture instead of snow.

Limitations of Common SE Practices
The International Council on Systems Engineering (INCOSE) defines system engineering as

" an interdisciplinary approach and means to enable the realization of successful systems."

This definition suggests that usability, being a discipline required for the realization of successful systems, is part of system engineering. However, apparently, common SE practices fail to prevent mishaps as those described above. The reason for this is that common SE practices persistently ignore the most critical system component, namely, the human operators (Case, 1997). Typically, we use the wrong measures. We measure elementary system attributes, but we should measure operator task attributes instead. To evaluate the impact of the human operator on the system utility, we need to evaluate the barriers to system efficiency and reliability. Research on Human Factors suggests that focusing on system performance and reliability is practically useless, unless we also take special care of the user performance and reliability:

- **Performance.** The time required for the operators to evaluate the system state and decide what to do next is typically higher by an order of magnitude than the system response time. Instead of measuring the system response time, we should measure the time elapsed from the moment the user decides to perform a task until its completion. Typically, most of the elapsed time is wasted because the user fails to follow the operational procedures, attempting to recover from unwanted system response to unexpected actions. SE should regard user productivity, rather than system performance (Landauer, 1993).
- **Reliability.** The operators MTBF is about 10% of the overall operation time, higher by several orders of magnitude than that of the system. Instead of measuring component failure rates, such as by MTBF, we should measure operational failure rates, such as the rate of almost-accidents due to user errors. This is especially true for safety-critical systems, in which the costs of an accident are much higher than those of maintenance. Operational reliability is the key to task performance. (Example: http://www.jnd.org/dn.mss/commentary_huma.html).
- **Recovery costs.** The operators' MTTR is about 50% of the overall operation time, higher by several orders of magnitude than that of the system. Instead of measuring maintenance costs, such as by MTTR, we should measure the time it takes for the users to recover from system failures.
- **Logic.** An application that is logical in its internal design and produces accurate results may nevertheless be difficult to use. The reason for this is that logic is not absolute. It is subjective, it is task related, and it changes over time. Typically, it applies to the internals of the application. Therefore, the user has difficulty following the developer's logic. (Buie and Vallone: http://www.aesthetic-images.com/ebuie/larger_vision.html).

Sources of the usability gap
Everybody in the system development team expects that it will be usable. Yet, it rarely happens. Berkun (http://www.scottberkun.com/essays/22-the-list-of-reasons-ease-of-use-doesnt-happen-on-engineering-projects/) provided a list for why systems are not always easy to use. This section provides an overview of the forces within the system development team that act against usability, and proposes that customer utility should be set as the main goal. It should be commented that naturally, many system engineers deny the kind of critics that this section might hinder (one of the reviewers commented that the article overuses cartoons as sources).

- **Technology-oriented engineering.** SE is governed by technology, rather than by customer needs. Typically, developers (e.g., programmers) who are fond of technology are careless about user needs (Weinberg, 1971; See also Dilbert cartoon: http://web.mit.edu/is/usability/IAP/2003/Session1/Images/reboot.gif). Many system engineers who are aware of the significance of the operator's attributes generalize them to define intelligent systems, (e.g., Oliver et al., 1997), thereof disregarding critical properties of the human operator.
- **The developer's intuition.** An intuitive interface asks no more of the user than what they either already know, or can immediately deduce from previous life experience. Implied is that intuition is wisdom assumed and shared within a community — the community of users familiar with the task and with the environment in which it is performed (Buie and Vallone, 1997). The usability problem results from the developers' intuition, that of highly skilled users, being applied to regular users, who are not familiar with the system behavior (Martin cartoon: http://www.nevtron.si/borderline/archive2/intuiti.gif). After getting used to the prototype, developers typically judge the system behavior as experienced users (e.g., Dilbert cartoon:

http://web.mit.edu/is/usability/IAP/2003/Session1/Images/ctrl-alt.gif). For them, the system behavior is obvious, and they fail to understand why a user, who sees a certain feature for the first time, would not realize what it should do, and how (e.g. Dilbert cartoon: http://web.mit.edu/is/usability/IAP/2003/Session1/Images/Stupid-users.gif).

- **Designers Creativity.** UI designers do not always promote usability: simple UI appearance, easy look and feel might often be boring for some designers. For example, website designers love to apply flash technology, which is 99% bad (Nielsen alertbox: http://www.useit.com/alertbox/20001029.html).

The Usability Gap

What is missing in traditional SE? The following table summarizes usability considerations typically missing in major SE disciplines:

SE Practice	**Common SE Practice**	**Typical Usability Gap**
System analysis	Excessive features satisfying marketing demands.	Users are slow. They fail to find the feature they need in time
System specification	Using SysML features	Popular SysML features hamper usability
Event-response definition	By use-cases	Enables user errors resulting in system failure: • Events that are unexpected and unacceptable in certain system states • System states that do not match the operational procedures
State definition	By state charts	Encourage mode-dependent system behavior, which enables mode errors
System architecture	Limited set of master requests	• Unlimited opportunities for user errors. • The system fails due to user errors
Interaction analysis	This is an informal activity. Operational procedures remain undefined. Users are expected to know the rules, although these are not defined yet.	Users unable to find the features they need, they do not know which option to select and what values to set.
Interaction specification and design	Informal based on use-cases, or by rapid prototyping by software experts	• Users do not follow the developer's intentions • Implements the error-prone mode-dependency
UI design	Attractors, such as animation, based on availability.	• Users distracted from their intentions • Users struggle to get their needs
Risk analysis	We focus on preventing system failures.	The system is not protected well against user errors. Mainly, the problem is that users fail to follow the system modes. (http://en.wikipedia.org/wiki/Mode_error)
System failure protection	We protect against expected failures.	• Users do not perceive the failure situation • Users do not recognize the system state • Users do not know how to resolve the problem
Error prevention	We assume users operate according to our intentions or instructions. We assume that users do not make mistakes and do not err	• Users model of the system is different from ours • Users make errors (Martin's cartoon: http://www.nevtron.si/borderline/delete.gif), often resulting in system failure.
User failure protection	We protect from risky events and from risky states	User events are sometimes unexpected, resulting in unexpected risky system states.
Testing	We assume that the users follow the (often undocumented) operational instructions	• Users operate not as presumed • The system does not handle unexpected user events • Critically risky unexpected user events not identified

Potential risk treatments
Once risks have been identified and assessed, all techniques to manage the risk fall into one or more of these four major categories (Dorfman, 2007):

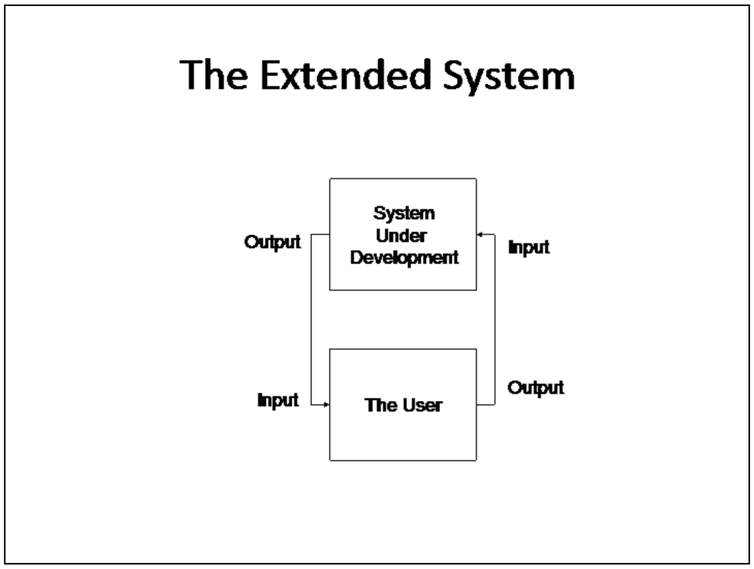
- Avoidance (eliminate)
- Reduction (mitigate)
- Transference (outsource or insure)
- Retention (accept and budget)

Barriers to risk treatments
In order to apply the treatments we need to have the management support in adopting a new strategy, which often contradicts the traditional strategies:

- **Marketing-oriented engineering.** People often confuse usability with marketing. However, marketing needs often conflict with usability (Dilbert cartoon: http://www.guuui.com/images/20030209.gif). The problem is that marketing follows the user's buying forces, which are different from their usability needs. For example, when applying banners in marketing campaigns, we intentionally distract the users from their original goals, in favor of the marketing goals. Marketing managers think of attracting potential customers, disregarding the actual customers (Dilbert cartoon: http://www.idblog.org/images/dilbert6-1.gif). They encourage usage of gimmicks, such as splash screens, to highlight new features that sell, regardless of the facts that these gimmicks hamper seamless operation. Marketing forces are according to the customers' wills, which are different from the users' needs. For example, a key feature that ensures usability is simplicity. However, marketing managers encourage complexity (http://www.joelonsoftware.com/items/2006/12/09.html). Leading usability practitioners have already noticed that people are not willing to pay for a system that looks simpler, because it looks less capable. Even a fully automatic system should contain lots of buttons and knobs, to make it look powerful (http://www.jnd.org/dn.mss/simplicity_is_highly.html). Before using a product, people will judge its desirability and quality based on 'what it does' (i.e. the number of features). Even though they may be aware that usability is likely to suffer, they will mostly choose products with many features. After having used these products however, usability will start to matter more than features and people will choose easy-to-use products over products with many features. The dilemma is that in order to maximize initial sales one needs to build products with many features, products that do lots of "stuff". But in order to maximize repeat sales, customer satisfaction and retention, one needs to prioritize ease-of-use over features (http://www.lukew.com/ff/entry.asp?433).
- **Customer-oriented engineering**. By disregarding usability, marketing managers often encourage developing systems that are difficult to use. Sometimes, however, they are right in doing so, because they do what the customers want, which is often not what they need. How should we balance usability against marketing? How can we conclude which of the two factors is more significant? The answer depends on the utility for the customers. However, even when marketing is considered more important, usability should be considered. For example, suppose that in order that the system looks powerful, the customers demand many features, and that all of them are apparent and easy to access. Still, usability engineering may enforce virtual simplicity, by highlighting the essential features and by separating them from the nice-to-have features.

Usability Engineering – UE
Usability engineering is the discipline for assuring the system's usability. Usability engineering implements human factors throughout the various disciplines involved in system engineering, to ensure that the system operation is fluent, efficient, reliable and safe. It is a cost-effective, user-centered process that ensures a high level of effectiveness, efficiency, and safety in complex interactive systems. Usability engineering is a structured, iterative, stepwise development process. Like the related disciplines of software and systems engineering, usability engineering is a combination of management principals and techniques, formal and semiformal evaluation techniques, and computerized tools.

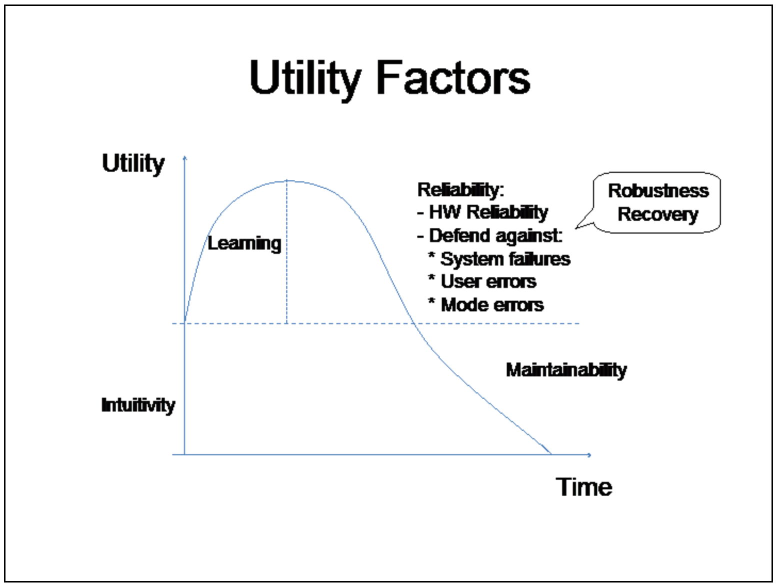
The Extended System
The extended system describes the customers' view of the system. Typically, the customers may be interested in technical features and in functional features, such as performance and reliability, but eventually, they need to know if the users can complete their tasks in time, how reliably they do their jobs and what are the safety levels involved. Therefore, besides the system under development, the extended system includes also the user and the user interaction with the system, as demonstrated in the following chart:



Typically, the user is the critical component of the extended system, implying the need for User-centered Design (UCD).

The Human Factors
The Usability Professional Association (UPA) defines usability as the degree to which something - software, hardware or anything else - is easy to use and a good fit for the people who use it (http://upassoc.org/usability_resources/about_usability/index.html) . However, ESE methodology is about another aspect of usability, namely, the human factors affecting the system utility.

Utility Assurance
The ESE methodology was developed to maximize the customer’s utility in the long run. The utility function can be described as in the following chart:



The utility function has two phases: The startup and the main phase. The startup phase begins with the initial usage of the system and ends with the utility function reaching maximum utility. The initial value of the utility function is determined by the intuitivity of the user interface. The slope from the initial value to the maximum value is determined by the ease of learning. Following the startup phase is the beginning of the main phase, in which the utility function stabilizes. Then the system utility gradually decreases. The reasons for utility decrease include hardware reliability and maintenance costs, well know in common QA. Additional reasons for the decrease of the system utility may be attributed to human factors, such as user errors and the user's capability to handle system failures.

The Need to Extend the System Engineering

Traditionally, usability engineering focuses on the system intuitivity and ease of learning, which are features of the startup phase. Eventually, common usability practices are adequate to deal with these aspects and are of low value when dealing with the main phase of the utility function. Common QA practices on the other hand, are applicable to the main phase. However, they focus on technical aspects of the system, disregarding the user's role. The human factors that affect the main phase of the system utility are not considered by any of the common practices. This is why and where we need to extend the system engineering.

Sources of User Difficulties

The ESE methodology considers two sources of user difficulties:

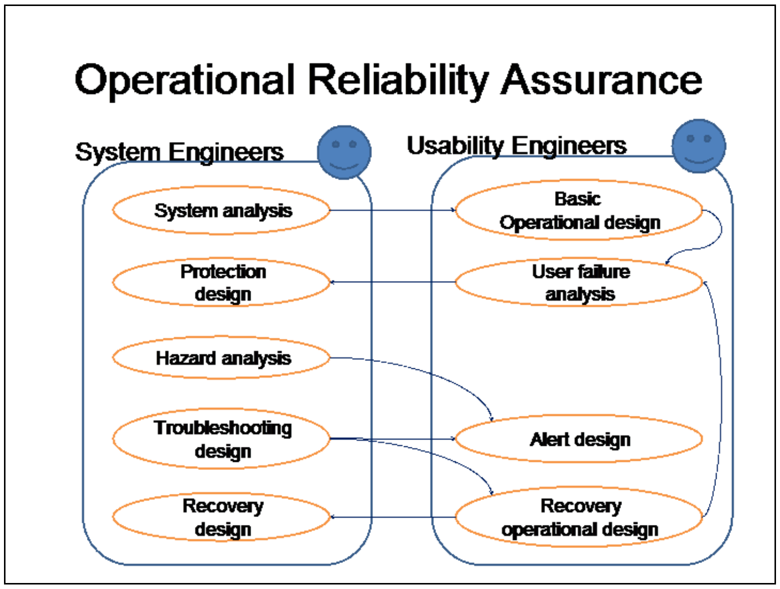
- User errors
- User incapability to handle system failures.

An example of an accident due to a user error is the ecological disaster of 1967 caused by the Torrey Canyon supertanker (http://en.wikipedia.org/wiki/Torrey_Canyon). The accident was due to a combination of several exceptional events, the result of which was that the supertanker was heading directly to the rocks. At that point, the captain failed to change the course because the steering control lever was inadvertently set to the Control position, which disconnected the rudder from the wheel at the helm (Casey, 1998).

Examples of the second type are the TMI accident described above, the NYC blackout following a storm (http://en.wikipedia.org/wiki/New_York_City_blackout_of_1977) and the chemical plant disaster in Bhopal, India (http://en.wikipedia.org/wiki/Bhopal_Disaster).

A Multi-disciplinary Approach

In order to define the extended system we need to integrate system engineering, which considers the technical factors, with human factors, which considers the user's limitations, and with usability engineering, which considers the user's behavior. The following chart describes the information exchange between the system engineers and the usability engineers:



Incorporating Human Factors in the Development Procedure

ESE methodology is based on iterative design (http://en.wikipedia.org/wiki/Iterative_design). In traditional SE, the iterations enable changes in the specifications and design during the system testing. In ESE, they enable early changes through prototyping (http://en.wikipedia.org/wiki/Prototyping) and late changes following usability testing.

- **System analysis.** Because the user is the critical component of the extended system, the system analysis should focus on optimizing the user's tasks and needs. Excessive functions, typically required by marketing, should be avoided (Dilbert cartoon: http://web.mit.edu/is/usability/IAP/2003/Session1/Images/Easy-to-use.gif). Tasks should be classified as either primary or secondary. Primary tasks should be highlighted. Redundant tasks should be removed (Functional simplicity: http://en.wikipedia.org/wiki/Occam's_Razor) or otherwise (if marketing insists that this is required) they should be isolated
- **System specification.** The common language used for system description is by SysML. This language uses various methods and tools for describing the system functions and behavior. The problem is that many of these methods and tools are not adequate for describing the extended system; the resulting system is beyond the user's capability. Following are two examples:
  - **Event-response definition.** Typical SysML-based event-response definition is by use-cases, which are derived at the analysis stage from scenarios. The problem is that such specification enables user events that are unexpected and unacceptable in certain system states, and also enables system states that do not match the operational procedures, which might result in system failure. In ESE methodology, beside use cases, the scenarios are also used to describe the operational procedures.
  - **State definition.** State charts facilitate the description of state dependency, resulting in excessive dependencies in the system definition. Consequently, the transition of user's actions to commands is state dependent, enabling mode errors.
- **System architecture**. In a system that integrates services from modules, such as in SOA, the client is responsible for selecting the proper function at the proper system state. When the client is a system unit, the selection procedure is predefined by the designers. However, when the client is a human operator, the selection sequence is fuzzy, and the consequences are unpredictable. ESE methodology provides guidelines to prevent such situations:
  - **A service interface**. A user interface that guides the user in the feature selection.
  - **Default behavior**. Consistent response to the user actions prevents operational ambiguity
  - **Safety net**. Model-based protocols enable detecting deviations from the pre-defined procedures.

Interaction analysis

Typically, there is no such a formal activity in the development cycle. ESE methodology includes a chapter on interaction analysis, including the following stages:

1. Defining a list of scenarios and user tasks, classify primary and secondary tasks
2. For each task, defining a context, set of assumptions, regarded as prerequisites
3. For each task, defining dependencies, such as data flow and sequencing
4. Complexity redundancy: removing redundant features and states.

Interaction specification and design

Typically, there is no such a formal activity in the development cycle. Typical designs implement use cases. When the design is based on use-cases, the users need to learn and remember which option is relevant to the current task. Therefore, such design is error-prone. ESE methodology includes a chapter on interaction specification, including guidelines for:

- Specifying operational procedures implementing the user primary tasks, for all scenarios
- Highlighting error-prone (state dependent) activity
- Specifying the means to prevent user errors
- Specifying the means to protect from expected user errors
- Specifying the means to capture and notify on unexpected user errors.

UI design

ESE methodology includes references to standard about user-centered design:

- Task-oriented animation
- Activity-oriented control definition.

Risk analysis

Common SE practices do not deal with user-related risks:

- **Error prevention**. SE should prevent all possible user errors, but they do not know when and how users might fail. User engineers should analyze the user errors, so that SEs can know what errors to prevent, and how.

- **Failure identification**. The problem of failure protection is that we expect usability engineers to handle failure situations, but we typically do not provide them with the information about when and how the system might fail. Failure protection is based on analysis of the system failures, so that usability engineers can consider the implication for the UI design.
- **Protection against unexpected user errors**. In ESE we include a safety network, which is a default event-response definition, enabling us to trap risky user events.

Usability-oriented testing

When including the user in the system, we need to conduct two types of usability testing:

- Traditional usability testing, intended to identify user difficulties during the startup phase. These may be conducted in special, well equipped usability labs, enabling recording the user activity and observing the users through a one-way mirror. (e,g, Dumas and Redish, 1999). Other forms of usability testing have been developed, (e.g., http://www.ergolabs.com/discount_usabilty_engineer.htm)which are more adequate for small-budget projects)
- Operational usability testing, intended to identify user failures during the system operation, at the integration testing stage or after the system has been deployed.

Common SE practices assume that the user knows and remembers all the operational instructions and the effect of activating each of the functions under all possible choices of options. This illusion of system designers blows in the face of testers who conduct usability testing, but usability testing can reveal only a small subset of the enormous number of possible failures. Common SE practices include verification that the system resists improper user activity, but they do not provide guidelines about how to define and present all possible exceptional user actions. ESE methodology includes guidelines for classifying the exceptional user events and how to include them in the testing.

Conclusion

ESE enables us to make sure that the users not only use the system according to the specification, but also according to the customer's expectation. In particular, the ESE methodology presented here enables us to avoid user confusion and to defend the system from exceptional user events.

Bibliography

1. Buie, E., A., and Vallone, A. Integrating HCI engineering with software engineering: A call to a larger vision. In Smith, M. J., Salvendy, G., & Koubek, R. J. (Eds.), *Design of Computing Systems: Social and Ergonomic Considerations* (Proceedings of the Seventh International Conference on Human-Computer Interaction), Volume 2. Amsterdam, the Netherlands: Elsevier Science Publishers, 1997, pp. 525-530.
2. Case, S. E. Towards user-centered software engineering. *Proceedings of Usability Engineering 2: Measurement and Methods (UE2)*. Gaithersburg, MD, March, 1997, tbd pages.
3. Casey, S. *"Set Phasers on Stun"*, Aegean Publishing: Santa Barbara, 1998
4. Dorfman, M., S. *Introduction to Risk Management and Insurance (9th Edition)*. Englewood Cliffs, N.J: Prentice Hall. ISBN 0-13-224227-3. 2007
5. Dumas, J.S. and Redish, J.C., *"A Practical Guide to Usability Testing"*, Exeter, England; Portland, Or.: Intellect Books, 1999
6. Landawer, T.K., *"The Trouble with Computers: Usefulness, Usability, and Productivity"*, MIT Press, 1993
7. Leventhal, L., and Barnes J., *Usability Engineering, Process, Products & Examples*, Pearson Education, Inc., Pearson Prentice Hall, 2008.
8. Paech, B., and Kohler, K., "Usability Engineering integrated with Requirements Engineering" ICSE Workshop *"Bridging the Gap between Software Engineering and Human-Computer Interaction"* 2003
9. Zonnenshein, A. & Harel, A, "Extended System Engineering – ESE: Integrating Usability Engineering in System Engineering". Poster presented at Incose International Symposium, Utrecht, The Netherlands. http://www.ergolight-sw.com/CHI/Company/Articles/ESE-Incose2008-P192.pdf