

## AN ANALYSIS OF DIFFERENT METHODS FOR WRITING HUMAN FACTORS REQUIREMENTS

Kenneth R. Allendoerfer  
Federal Aviation Administration  
William J. Hughes Technical Center  
Atlantic City International Airport, NJ

In software engineering, human factors (HF) and usability requirements have acquired a reputation for being ambiguous and hard to test. There are numerous styles in which HF requirements can be written, such as conforming to guidelines, human performance criteria such as speed and accuracy, satisfaction, and quality of output. Each of these methods has its own benefits and drawbacks, and projects must ensure that the style matches the goals and constraints of the project. In this paper, I give examples of each HF requirement style and describe the circumstances under which that style can be used effectively. I use examples from systems developed at the Federal Aviation Administration William J. Hughes Technical Center to discuss each style.

### INTRODUCTION

Authors in human factors (HF) often discuss the importance of usability to the success of interactive systems (Abran, Khelifi, Suryan, & Seffah, 2003; Schaffer, 2004; Vredenburg, Isensee, Righi, 2002). Their advocacy of user-centered design and usability testing has influenced some authors in the software engineering literature who consider usability “as one of the top criteria for commercial success and/or successful user adoption” (Leffingwell & Widrig, 2003, p. 259).

However, when it comes to writing formal requirements for use by software engineers, HF has acquired a bad reputation. Leffingwell and Widrig (2003) suggest that usability requirements can be “fuzzy” and that they present “a formidable challenge for the requirements team.” (p. 260). The Institute of Electrical and Electronics Engineers (IEEE) uses specifications that require a “good human interface” as an example of how not to write a specification (IEEE, 1998, p. 7). Even authors in HF acknowledge that some aspects of usability are not easy to express as formal requirements (Preece, Rogers, & Sharp, 2002).

In this paper, I discuss the literature on these topics and offer recommendations for writing better HF requirements. Throughout the paper, I use examples from Air Traffic Control (ATC) and Technical Operations (TechOps) systems that we have developed at the Federal Aviation Administration William J. Hughes Technical Center. ATC is the business of ensuring the safety and efficiency of air traffic; TechOps is the business of ensuring that the equipment used by ATC is maintained and functioning properly.

#### Definition and constructs

HF requirements are usually considered part of a system’s nonfunctional requirements (Leffingwell & Widrig, 2003). That is, the HF requirements establish

constraints on how developers may meet the functional requirements that directly affect the human users. While I certainly agree with this, HF also should be concerned with functional aspects of the system. A system that has a beautiful user interface and follows every usability heuristic but lacks fundamental functions cannot be considered to have high usability.

A better, broader construct is offered by Bevan (1999) and the International Organization for Standardization (ISO) when defining their concept of quality in use: “the effectiveness, productivity and satisfaction of the user when carrying out representative tasks in a realistic working environment” (p. 90). In this definition, quality in use is a product of both functional and non-functional attributes. A system cannot achieve high quality in use unless it meets the needs of users, and a system cannot meet the users’ needs unless the necessary functions are available.

#### Characteristics of a good human factors specification

A good HF specification has the same attributes as any good specification. The IEEE (1998) characterizes a good specification as one that is correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, and traceable. Human factors engineers (HFEs) writing requirements, typically as a part of a requirements team, often have trouble with ambiguity and verification. HF requirements can be ambiguous because many terms used to describe HF attributes, words like easy, user friendly, and usable, have different meanings to different people and contexts (Abran, Khelifi, Suryan, & Seffah, 2003). HFEs must define and operationalize their terms for a given user group, task, and environment. This requires observation, discussion, and analysis; usually coupled with prototyping, testing, or modeling.

In addition to the ambiguity of terms, HF requirements can be very difficult to test. Users often do not express their needs in testable terms; they just request that the system be easy to use. Fortunately, well-defined terms in the specification will often suggest ways that the

requirements can be verified. Unfortunately, many software testers lack the expertise to conduct valid usability tests and are not prepared to develop new methods or consult with HFEs.

## STYLES OF HUMAN FACTORS AND USABILITY REQUIREMENTS

Lauesen and Younessi (1998) describe styles in which HF and usability requirements are typically written. In the sections that follow, I describe these and other styles and offer my analysis of each style's advantages and disadvantages. In particular, I discuss how the styles relate to ambiguity and verification because these are common ways that HF requirements are unsuccessful.

### Guidelines and standards

In this style, the specification requires that the system follow an existing user interface guideline or standard. In the FAA, the applicable standard is the *Human Factors Design Standard* (HFDS [Ahlstrom & Longo, 2003]). For an ATC system, a guidelines-based requirement might be "The system shall conform to Chapter 8 of the HFDS."

One benefit of this style is that following the guidelines will provide a base level of usability without significant new research and development costs. Software engineers are accustomed to following standards in other areas of development (e.g., coding standards, data standards) so requiring a user interface standard will match existing practices.

One drawback of this style is that guidelines, especially for consumer level software, can be overly general (Bevan & Macleod, 1994) and may apply to only common types of functionality. If the specification is being created for a system with unique functionality or a specialized purpose, industry standards may not be applicable or sufficient. HFEs should favor guidelines that are specific about what users, tasks, and environments they apply to. HFEs should favor guidelines that provide information about the implications of the guideline for performance and outcomes.

A second drawback is that a guideline document may contain many individual guidelines, only some of which apply to the system at hand. A specification requiring compliance with an entire guideline could force the testers to verify conformance to dozens of irrelevant guidelines. HFEs should plan to select or prioritize individual guidelines within a full guideline document.

Guideline-based requirements are most useful when the system contains well-understood functionality and when the system is being developed internally. When the system is being bought off-the-shelf, guidelines-based requirements may not be useful because vendors may have already built their products and may have used different standards or none at all (Lauesen and Younessi, 1998). Requiring that vendors follow a specific standard may eliminate otherwise good products from consideration.

### User interface design

In this style, the specification requires that the system appear and behave in a particular way. The specification describes or provides diagrams of menus, pages, interaction sequences, and so on. In ATC, a design-based requirement might be "The system shall provide the following pull-down menus on every controller window: Display, Maps, Flights, Alerts, and Weather."

The main benefit of design-based requirements is that they are easy to understand and test. The more detailed the specified design is, the less guessing the developers have to do. If the specification provides a screenshot and requires that the system conform to it, it is trivial for a tester to verify whether the system meets the requirement.

One drawback of this style is that it does not fit with system development processes that draw firm lines between requirements and design. In these processes, deciding how screens should appear is the job for developers, not a requirements team. Organizations that follow these processes discourage detailed designs from appearing in a requirements document.

A second drawback is that the design risk is assumed by the requirements team rather than by the developers. If the design ultimately fails, it is the requirements team's fault. This necessitates the team have a good understanding of what a good design would be before they write requirements. In addition, though some developers may be glad to shift the design risk to the requirements team, it removes some of the creative and rewarding aspects of developers' jobs.

When the system has known constraints, such as special environmental conditions or user abilities, design-based requirements are very useful. When performance requirements are unknown, specifying the design may be the only option. To ensure that the specified design is sound, HFEs should provide design rationales that explain why the requirements are how they are (Vredenburg, Isensee, Righi, 2002). Rationales can be based on best practices, standards, prototyping, testing, or modeling. Mandatory rationales force HFEs to consider the implications and tradeoffs of their designs and increase the likelihood of specifying a good design.

As with guidelines, this style does not fit well with projects where the system will be bought rather than built. Vendors will have already created products that may not conform to the specified design (Lauesen and Younessi, 1998).

### Human factors processes

In this style, the specification lists HF activities, such as prototyping and usability testing, that must be conducted during the system lifecycle. The specification should describe each activity and provide criteria for when an activity is considered complete. In ATC, a process-based HF requirement might be "The vendor shall conduct four

rapid prototyping sessions with 5 controllers each, during which feedback about the user interface shall be collected.”

The main advantage of process-based requirements is that HF activities become mandatory rather than “nice to have.” In an ideal world, these activities would already be codified in the organization’s development processes.

The main drawback of this style is that it may not be clear to the requirements team which HF activities are appropriate for the system being designed. Another concern is that following a good process does not guarantee a good product.

This style is most useful when the system is exploratory or not well-defined. Though there is no guarantee that the final system will have high usability, following good HF processes increases the odds.

### Human performance

In this style, users and tasks are specified with statistical criteria for success, typically in terms of speed, accuracy, number of steps, and so on. Some authors use the terms effectiveness and efficiency to capture similar concepts (Bevan, 1999). In TechOps, a performance-based HF requirement might be “The system shall allow technicians to complete the radar diagnostic procedure in an average of 10 minutes or less.”

Performance-based requirements can be expressed in absolute terms or relative to legacy systems or processes. In either case, the performance criteria should be derived from known performance levels. However, in my experience, organizations frequently do not have baseline data about current performance. Alternately, organizations may select performance criteria that sound good but have no basis in fact or are influenced by optimistic performance levels quoted by vendors.

Creating nonarbitrary and achievable performance criteria requires commitment from the project to allow HFEs to observe the users in their working environments and collect information about current systems and processes. It may also require some prototyping and testing during the development stage to see if developers are making progress toward meeting the performance requirements and if a change is necessary.

A second approach to nonarbitrary and achievable criteria is through cognitive modeling. If a model is available of users’ tasks and abilities, the effect of changes to the task or environment can be measured. This requires a commitment from management to creating and validating models and updating them when users, tasks, and technology change. Models can be very resource intensive and are usually only cost effective when the system is mission critical.

Another consideration of performance-based requirements is that they give little guidance to developers. Usual functional requirements give software engineers ideas for what the system should ultimately be like. However, most software engineers have little experience building to meet human performance requirements and may

not know where to begin. In the earlier example, would a software engineer know, based only the specification, how to build a system that would allow a technician to complete a procedure in 10 minutes or less?

In addition, developers may be reluctant to accept performance-based requirements, even if they are testable, because the tests are complex and expensive to run. They require recruiting participants, developing realistic scenarios, and using unfamiliar measurement techniques. In addition, usability testing is not easy to automate so many existing software testing methods and practices cannot be used.

Finally, human performance testing requires that a system be mature before human performance tests can occur. Software engineers are accustomed to conducting multiple, small-scale unit tests before integrating code with the larger system. When testing human performance, noninteractive prototypes and other so-called discount usability methods (the HF equivalents of unit tests) are not meaningful, especially when the human performance metric is speed. Each aspect of the system that affects the task should be available (or realistically simulated) to evaluate human performance.

Performance-based requirements may seem like a gold standard, but when the requirements team is very knowledgeable about the users, tasks, or environment, it may be simpler to use design-based requirements. For example, in TechOps, suppose the HFE on the requirements team knows that technicians often have trouble reading a font smaller than 8 point. Strict advocates for separating requirements and design would argue that the team should establish a readability criterion, require it in the specification, and have the designers determine what font size to use. However, the developers probably will not know which font size would meet the criterion so they will consult a user interface guideline and base their design on that. Effort could be saved if the team were simply allowed to specify 8-point font in the first place.

### Satisfaction

In this style, the specification requires that the system achieve a level of subjective satisfaction from its users. Some authors call this hedonic quality, to distinguish from ergonomic quality (Hassenzahl, Platz, Burmester, & Lehner, 2000). Satisfaction is especially important for consumer and publicly accessible systems where users can choose whether to use the system. Satisfaction requirements often have close ties to an organization’s public relations or employee morale goals. In ATC, a subjective satisfaction requirement might be “80% or more of the controllers shall rate the usability of the display as High or Very High.”

A specification including satisfaction requirements should define how satisfaction is to be measured, typically by rating scales. Because satisfaction can be changeable, a good practice is to require satisfaction measurements be taken at several points in time.

As with the performance style, satisfaction requirements do not provide developers much information about what to build. The requirements team should provide information to the developers about the features, user interface elements, and other attributes that are likely to achieve the required levels of satisfaction. To do this, prototyping, focus groups, and other similar activities may be necessary.

Requiring a level of satisfaction may be appealing as a shortcut to specifying and testing performance requirements. However, some authors have found that measures of performance and measures of satisfaction do not correlate well (Frøkjær, Hertzum, & Hornbæk, 2000).

### **Defect rate**

In this style, the specification sets a maximum number or severity for usability problems identified during testing. In ATC, a defect-rate HF requirement might be “No more than 10 moderate user interface problems and no more than 2 severe problems shall be identified during testing.”

To use this style, a taxonomy for usability defects must be established for the system. The taxonomy must be clear, complete, and designed to be consistently applied by different evaluators. Because evaluators may disagree on the defects and their severity, multiple independent evaluators should be used and an average or group consensus taken.

Defect rate requirements are common in software engineering and should be familiar to developers and testers. However, creating the taxonomy and the evaluation methods are significant HF projects by themselves. The taxonomy probably will be derived from existing guidelines and past projects and the defects identified using a common usability testing method.

### **Training**

Lauesen and Younessi (1998) consider training to be part of the performance style, but others (Vredenburg, Isensee, Righi, 2002) treat it as a separate construct. Training-based HF requirements identify how much time or experience is needed before users achieve a specified level of performance. In TechOps, a training-based requirement might be “A technician shall be able to complete the radar diagnostic procedure correctly after no more than two days using the system.”

One advantage of training-based requirements is that they have good face validity with stakeholders. Reducing training and transition costs is normally advantageous to the organization and is often used to justify usability improvements.

On the other hand, training-based requirements share many of the drawbacks of performance-based requirements. Developing nonarbitrary values for the training time frame necessitates that the team collect and analyze baseline data before requirements are written and long before the system

is available. Is the specified training time frame actually feasible?

### **Help requests**

This style combines elements of the training, defect, and satisfaction styles. Here the specification identifies a maximum number or rate of help requests made by users. A help request can be an indicator of a usability or training problem with the system. Help requests can be based on usage of online help or on calls to the help desk or customer support line. This HF requirement style can be written relative to current help request rates.

The main advantage of this style is that help desk support after deployment is a major contributor to total cost of ownership. Like reducing training costs, reducing help desk calls is very appealing to project management and can help justify the costs of the HF activities. This style of requirement is rooted in human performance and is fairly easy to quantify and measure (e.g., number of calls per month).

The main drawback of this style is that it is dependent on users' perceived quality of the available help and is affected by their attitudes regarding asking for help. If the users are reluctant to call the help desk because of previous bad experiences, the reduction in calls to the help desk is a poor indicator of how usable the system is.

### **Outcome**

Descriptions of performance requirements, such as those given by Lauesen and Younessi (1998) and others, usually focus on speed, accuracy, number of steps, and other related metrics. This is appropriate for many systems, especially those in which the same tasks are repeated frequently and accuracy is easy to determine. However, these metrics do not apply well to systems that support problem solving, decision-making, situation awareness, creativity, or other less structured tasks. Is being able to make a decision quickly the best metric available to measure the effectiveness of an ATC decision-support tool? How does accuracy apply to a system that helps an TechOps technicians identify long term trends or compose maintenance orders? The standard approach to writing human performance requirements does not capture quality in use for these systems. Another style, which I call outcome-based requirements, is necessary.

Outcome-based requirements specify a level of acceptable quality for the product that users generate when using the system. The requirement can be expressed in absolute terms or in terms of improvement over current outcomes. In ATC, an outcome-based requirement might be “The system shall increase the quality of the reroutes controllers develop.” Reroutes created using the system would need to be examined by an independent domain expert to determine if the requirement has been met. One way to do this would be to evaluate reroutes before

deployment and periodically evaluate them after deployment under similar circumstances.

For some systems, both performance requirements and outcome requirements can apply. For example, a highly usable maintenance tracking system would allow TechOps users to find information quickly (performance) and would also help the technicians improve the quality of maintenance service they provide (outcome).

The biggest drawback of outcome-based requirements is that they are extremely difficult to verify, especially before the system is available. Simulations are one way to test outcomes before deployment, but simulations need to be high fidelity to meaningfully measure outcomes. Another approach is to deploy the system to one facility as a field test and assess outcomes there before its full deployment.

This style is best for systems where simpler performance metrics like speed and accuracy are not very meaningful, such as systems with unstructured problems or tasks. HFEs writing outcome-based requirements need to understand what a high-quality outcome would be and how to measure it. To do this, definitions and preferably examples of good outcomes must be collected, and a systematic method for evaluating outcomes must be created.

### Functional requirements

Some functional requirements so significantly affect usability that they really are HF requirements. If critical functions are missing or do the wrong thing, the system cannot have high quality in use regardless of its other HF attributes. The recent focus in software engineering on use cases as part of the development process demonstrates the interest among engineers in developing functional requirements that are usable (Leffingwell & Widrig, 2003).

The functional requirements that directly affect users should be reviewed and tested by HFEs as well as the software testers. These usability tests should be organized around user tasks and workflow rather than around technical considerations.

When developing functional requirements, HFEs should do more than ask users what functions they want. They should observe users interacting with current systems and processes and determine which are frequent and critical. HFEs should develop prototypes that allow users to complete tasks using different combinations of functions. The functions the users *actually* select and use, rather than ones they *claim* they would use, should be the focus of further HF activities.

### CONCLUSIONS

Good HF requirements can be written in any of these styles. Complex sociotechnical systems like those developed by the FAA often will require that several styles be used in combination. HF will have the greatest impact

when practitioners are versed in each of these styles and can select the best ones for the project at hand.

In addition, HFEs and software engineers should adjust their attitudes and expectations for HF requirements. If HFEs are going to specify a design, performance, or outcome level as a requirement, they should be confident that the requirement is achievable and that it serves a clear user need. This means that projects must be willing to conduct or sponsor more HF testing before requirements are written to ensure that criteria are sound. This also requires that HFEs be willing to revise the criteria if they prove to be unachievable.

On the other hand, system developers need to demonstrate more willingness to accept tough HF requirements. I have worked with numerous developers who would not sign up for a performance- or outcome-based HF requirement if it were put to them. In their view, doing so would be to accept too much risk. If systems are ever to achieve high quality in use, developers need to think of HF requirements like system uptime or other rigorous requirements: as a challenge to be overcome.

### REFERENCES

- Abran, A., Khelifi, A., Suryan, W., & Seffah, A. (2003). Usability meanings and interpretations in ISO standards. *Software Quality Journal, 11*, 325–338.
- Ahlstrom, V., & Longo, K. (2003). *Human Factors Design Standard (HF-STD-001)*. Atlantic City International Airport: Federal Aviation Administration William J. Hughes Technical Center.
- Bevan, N. (1999). Quality in use: Meeting user needs for quality. *Journal of Systems and Software, 49*, 89–96.
- Frøkjær, E., Hertzum, M., & Hornbæk, K. (2000). Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated?. *Proceedings of the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2000*, 345–352.
- Hassenzahl, M., Platz, A., Burmester, M., & Lehner, K. (2000). Hedonic and ergonomic quality aspects determine a software's appeal. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2000*, 201–208.
- Institute of Electrical and Electronics Engineers (IEEE). (1998). *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*. New York, NY: Author.
- Lauesen, S., & Younessi, H. (1998). Six styles for usability requirements. *Proceedings of the International Workshop on Requirements Engineering (REFSQ), 4*, 155–166.
- Leffingwell, D., & Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach* 2nd Edition. Boston, MA: Addison-Wesley.
- Preece, J., Rogers, Y., & Sharp, H. (2002). *Interaction Design*. New York, NY: Wiley.
- Schaffer, E. (2004). *Institutionalization of Usability: A Step-by-Step Guide*. Boston, MA: Addison-Wesley.
- Vredenburg, K., Isensee, S., Righi, C. (2002). *User-Centered Design: An Integrated Approach*. Upper Saddle River, NJ: Prentice Hall.