



INSTITUTE FOR DEFENSE ANALYSES

**Users Are Part of the System:
How to Account for Human Factors When
Designing Operational Tests for Software Systems**

Laura J. Freeman, *Project Leader*
Kelly M. Avery
Heather M. Wojton

July 2017

Approved for public release;
distribution is unlimited.

IDA NS D-8630

Log: H 2017-0000426

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882



The Institute for Defense Analyses is a non-profit corporation that operates three federally funded research and development centers to provide objective analyses of national security issues, particularly those requiring scientific and technical expertise, and conduct related research on other national challenges.

About This Publication

The Director, Operational Test and Evaluation (DOT&E) has issued several policy memos emphasizing statistical rigor in test planning and data analysis, including the use of design of experiments (DOE) principles, and smart survey design and administration. Oftentimes, particularly when testing software-intensive systems, it is necessary to account for both engineering and human factors simultaneously in order to facilitate a complete and operationally realistic evaluation of the system. While some software systems may inherently be deterministic in nature, once placed in their intended environment with error-prone humans and highly stochastic networks, variability in outcomes can, and often does, occur. This talk will briefly discuss best practices and design options for including the user in the DOE, and present a real-world example.

Acknowledgments

Technical review was performed by Laura J. Freeman and Matthew R. Avery from the Operational Evaluation Division

For more information:

Laura J. Freeman, Project Leader
lfreeman@ida.org • (703) 845-2084

Robert R. Soule, Director, Operational Evaluation Division
rsoule@ida.org • (703) 845-2482

Copyright Notice

© 2017 Institute for Defense Analyses
4850 Mark Center Drive, Alexandria, Virginia 22311-1882 • (703) 845-2000.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (a)(16) [Jun 2013].

INSTITUTE FOR DEFENSE ANALYSES

IDA NS D-8630

**Users Are Part of the System:
How to Account for Human Factors When
Designing Operational Tests for Software Systems**

Laura J. Freeman, *Project Leader*

Kelly M. Avery

Heather M. Wojton

Executive Summary

The goal of operation testing (OT) is to evaluate the effectiveness and suitability of military systems for use by trained military users in operationally realistic environments. Operators perform missions and make systems function. Thus, adequate OT must assess not only system performance and technical capability across the operational space, but also the quality of human-system interactions.

Software systems in particular pose a unique challenge to testers. While some software systems may inherently be deterministic in nature, once placed in their intended environment with error-prone humans and highly stochastic networks, variability in outcomes often occurs, so tests often need to account for both “bug” finding and characterizing variability.

This document outlines common statistical techniques for planning tests of system performance for software systems, and then discusses how testers might integrate human-system interaction metrics into that design and evaluation.

System Performance

Before deciding what class of statistical design techniques to apply, testers should consider whether the system under test is deterministic (repeating a process with

the same inputs always produces the same output) or stochastic (even if the inputs are fixed, repeating the process again could produce a different result).

Software systems—a calculator, for example— may intuitively be deterministic, and as standalone entities in a pristine environment, they are. However, there are other sources of variation to consider when testing such a system in an operational environment with an intended user. If the calculator is intended to be used by scientists in Antarctica, temperature, lighting conditions, and user clothing such as gloves all could affect the users’ ability to operate the system.

Combinatorial covering arrays can cover a large input space extremely efficiently and are useful for conducting functionality checks of a complex system. However, several assumptions must be met in order for testers to benefit from combinatorial designs. The system must be fully deterministic, the response variable of interest must be binary (pass/fail), and the primary goal of the test must be to find problems. Combinatorial designs cannot determine cause and effect and are not designed to detect or quantify uncertainty or variability in responses.

In operational testing, the assumptions listed above typically are not met. Any number of factors, including the human user, the network load, memory leaks, database

errors, and a constantly changing environment can cause variability in the mission-level outcome of interest. While combinatorial designs can be useful for bug checking, they typically are not sufficient for OT. One goal of OT should be to characterize system performance across the space.

The appropriate designs to support characterization are classical or optimal designs. These designs, including factorial, fractional factorial, response surface, and D-optimal constructs, have the ability to quantify variability in outcomes and attribute changes in response to specific factors or factor interactions.

These two broad classes of design (combinatorial and classical) can be merged in order to serve both goals, finding problems and characterizing performance. Testers can develop a “hybrid” design by first building a combinatorial covering array across all factors, and then adding the necessary runs to support a D-optimal design, for example. This allows testers to efficiently detect any remaining “bugs” in the software, while also quantifying variability and supporting statistical regression analysis of the data.

Human-System Interaction

It is not sufficient only to assess technical performance when testing software systems. Systems that account for human factors (operators’ physical and psychological characteristics) are more likely to fulfill their missions. Software that is psychologically challenging often leads to mistakes, inefficiencies, and safety concerns.

Testers can use human-system interaction (HSI) metrics to capture software compatibility with key psychological characteristics. Inherent characteristics such as short- and

long-term memory processes, capacity for attention, and cognitive load are directly related to measurable constructs such as usability, workload, and task error rates.

To evaluate HSI, testers can use either behavioral metrics (e.g. error rates, completion times, speech/facial expressions) or self-report metrics (surveys and interviews). Though behavioral metrics are generally preferred since they are directly observable, the method you choose depends on the HSI concept you want to measure, your test design, and operational constraints.

The same logic can be applied to HSI data collection as data collection for system performance. Testers should strive to understand how users’ experience of the system shifts with the operational environment, thus designed experiments with factors and levels should be applied. In addition, understanding if, or how much, user experience affects system performance is key to a thorough evaluation.

The easiest way to fit HSI into OT is to leverage the existing test design. First, identify the subset (or possibly superset) of factors that are likely to shape how users experience the system, then distribute those users across the test conditions logically. The number of users, their groupings, and how they will be spread across the factor space all matter when designing an adequate test for HSI.

Most HSI data, including behavioral metrics and empirically validated surveys, also can be analyzed in the same way system performance data can, using statistically rigorous techniques such as regression. Operational conditions, user type, and system characteristics all can affect HSI, so it is critical to account for those factors in the design and analysis.



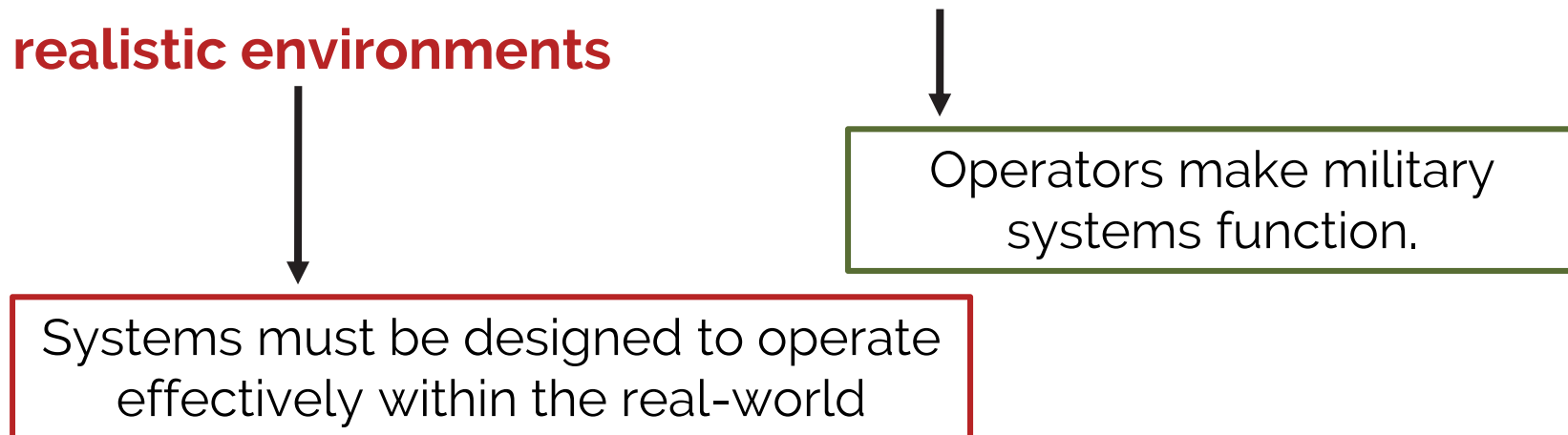
Users are Part of the System:
How to Account for Human Factors When
Designing Operational Tests for Software Systems

Kelly McGinnity Avery
Heather Wojton
Institute for Defense Analyses

July 31, 2017

Operational tests include the system and context

OT Goal: Test and evaluate the effectiveness and suitability of military systems for use by **military users** in **operationally realistic environments**



Adequate operational tests must address:

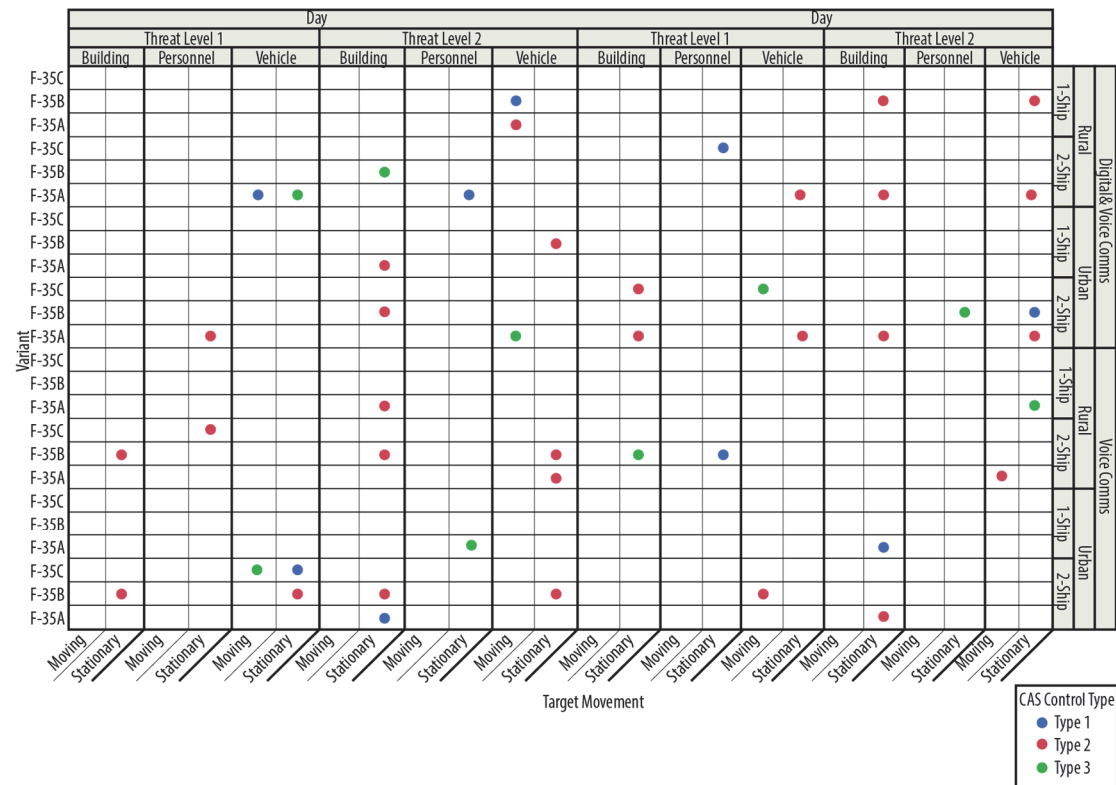
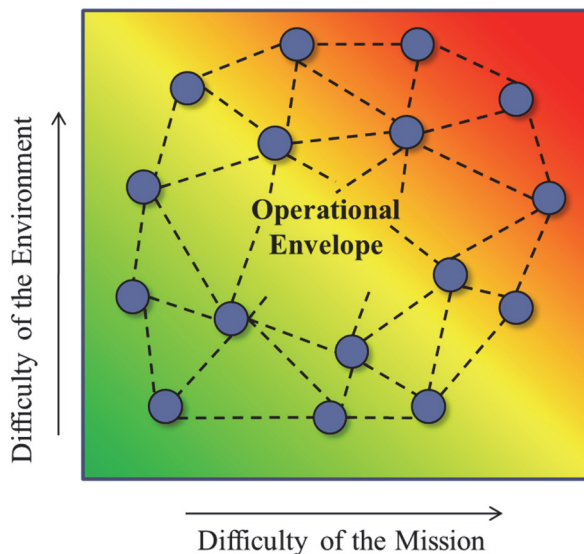
- System performance and technical capability across the operational space
- Quality of human-system interactions

System Performance

Design of Experiments (DOE) techniques have become standard practice for hardware-centric systems

DOE provides a scientific, structured, objective test methodology answering the key questions of test:

- How many points?
- Which points?
- In what order?
- How to analyze?



What about software systems? Does DOE still apply?

Yes! Though practitioners should think carefully about **variability** in their outcome of interest as this affects the type of design that is appropriate



Two broad categories:

- Deterministic ← Repeating a process with the same inputs always produces the same output
- Stochastic ← Even if the inputs are fixed, repeating the process again could change the outcome

Notional example: Testing a new calculator

Goal is to test the **accuracy of basic operations** on your calculator (e.g. when a user types 3×4 does the calculator display 12?)



Is this outcome is deterministic or stochastic?



The twist: The new calculator is intended for **scientists** to use in **Antarctica**

Incorporating the user and the operational context

What **sources of variation** should be considered when testing the system in its operational environment with the intended user?

- User Clothing (gloves vs. not)
- Temperature
- Lighting
- ...



We should fully evaluate calculator functionality *along with* these factors of interest

Combinatorial designs are useful for functionality checks...

Combinatorial covering arrays can cover a large input space extremely efficiently!

Criteria for using combinatorial designs:

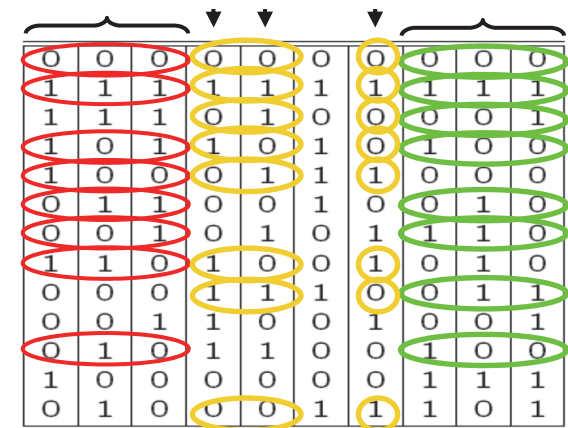
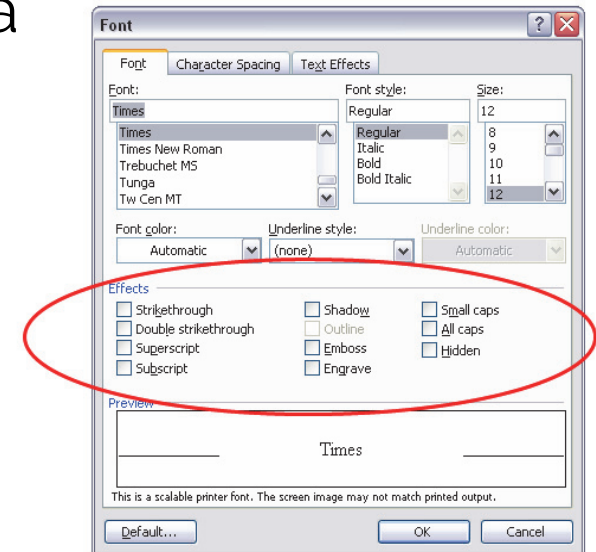
- System is fully deterministic
- Primary goal is to find problems
- Pass/fail response

Limitations:

Cannot determine cause and effect

Ignores / does not quantify variability

Microsoft Word example



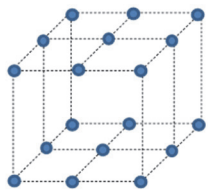
...but typically aren't sufficient (or even appropriate) in an operational context!

Testers need to account for variability caused by:

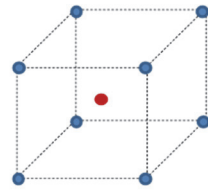
- Human users!
- Network load
- Database errors
- Dynamic Environment
- Memory Leaks
- ...

Remember the penguin!

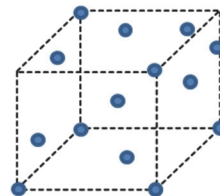
Classical/optimal designs are more appropriate for this purpose



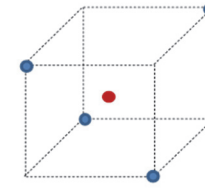
General Factorial
3x3x2 design



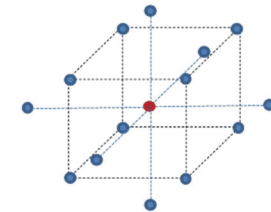
2-level Factorial
 2^3 design



Optimal Design
IV-optimal



Fractional Factorial
 2^{3-1} design



Response Surface
Central Composite design

A hybrid design can address the goals of both classes of DOE without dramatically increasing resources

Combinatorial designs are intended to find problems

Classical designs are intended to characterize performance across a set of factors

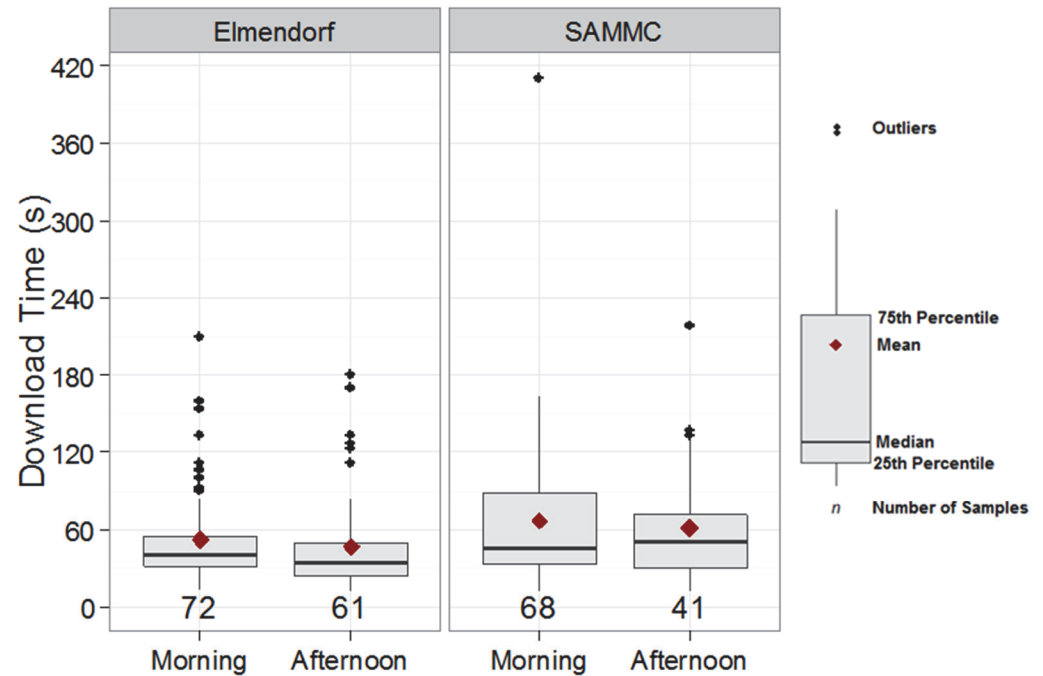
Both are reasonable goals when testing software in a complex environment

“Augment up” approach:

1. Build a base combinatorial design
2. Add the necessary runs to support a statistical model

Facilitates coverage of the space AND meaningful statistical analysis

C #	W 1	W 2	W 3	W 4	W 5	W 6	W 7	W 8	W 9	W 10	W 11	W 12	W 13	W 14	W 15	W 16	W 17	W 18	W 19	W 20	W 21	W 22
1	0	0	0	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	1	1	0	1	1	1	1	0	1	0	0	0	0	0	1
3	0	1	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	1	0	0	0	1
4	0	1	1	0	0	0	1	1	0	0	1	0	1	0	1	0	1	0	0	0	0	1
5	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	1	1
6	1	0	1	0	0	0	1	1	0	1	0	0	0	1	1	1	0	0	0	0	0	1
7	1	1	0	0	0	0	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1
8	1	1	1	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
9	0	0	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0
11	0	0	0	1	0	0	1	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0
12	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0



Human-System Interaction

Software systems that ignore operator psychology underperform

Systems that account for the human factor are more likely to fulfill their missions

The human factor includes operators' physical and psychological capabilities and characteristics

Difficulties with software systems are usually psychological

- Recalling the label for a particular command
- Finding commands because they are buried under counterintuitive menus or irrelevant data

For example, why do we have to select the *Start* menu to find the *Shut Down* command?

Software that is psychologically challenging leads to mistakes, inefficiencies, and safety concerns

Well-known psychological characteristics directly affect key HSI metrics

Inherent characteristics

Short- and long-term memory processes

Ability to perform complex mental calculations

Expectations for cause and effect, and meaning making

Capacity for pattern recognition

Capacity for attention

Cognitive load and capacity for multi-tasking

Metrics that can capture software compatibility with these characteristics

Usability

Workload

Task Error Rates

Workflow

Trust

Reliance

Several methods exist to evaluate HSI

Broadly, these methods are placed into 2 groups:

Behavior

- Human error rates
- Task completion time
- Ability to complete tasks
- Steps required to obtain info
- Speech or facial expressions

Self-Report

- Surveys
- Interviews

The method you choose depends on the HSI concept you want to measure, your test design, and operational constraints

Behavioral metrics are preferred because they are observable

Self-report metrics are used to supplement behavioral metrics or alone if it is not possible to capture a behavioral metric

Leverage The Existing Test Design

Apply the same logic to HSI data collection that you do to data collection for system performance

Collect the HSI data across the test space to:

1. Understand how users' experience of the system shifts with the operational environment
2. Understand how users' experience affects system performance



Identify factors that are likely to shape how operators experience the system

	Data Size Small				Data Size Large			
	Low Load		High Load		Low Load		High Load	
	New Patient	Existing Patient	New Patient	Existing Patient	New Patient	Existing Patient	New Patient	Existing Patient
Emergency Room	3	3	3	3	3	3	3	3
Intensive Care	3	3	3	3	3	3	3	3
Obstetrics	3	3	3	3	3	3	3	3
Ambulatory Medical	3	3	3	3	3	3	3	3
Ambulatory Surgical	3	3	3	3	3	3	3	3
Ambulatory Dental	3	3	3	3	3	3	3	3
Small Clinics	3	3	3	3	3	3	3	3

subset
→

	New Patient	Existing Patient
Emergency Room	12	12
Intensive Care	12	12
Obstetrics	12	12
Ambulatory Medical	12	12
Ambulatory Surgical	12	12
Ambulatory Dental	12	12
Small Clinics	12	12

Administer usability surveys across these factors

You may not need to identify a subset of factors if all factors are likely to affect how operators interact with the system

Distribute Users Across Test Conditions Logically

Identify the number of users

How users will be broken into groups

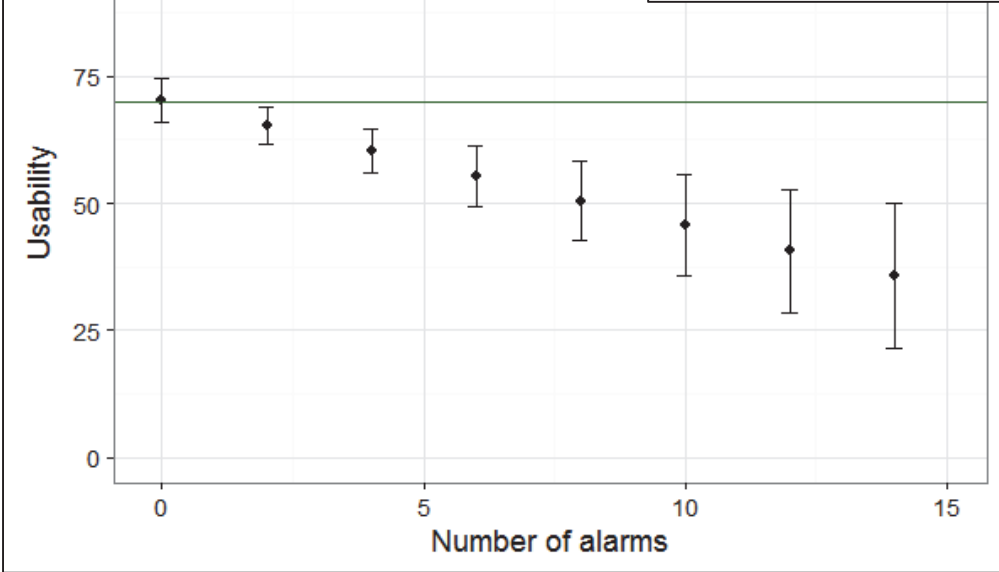
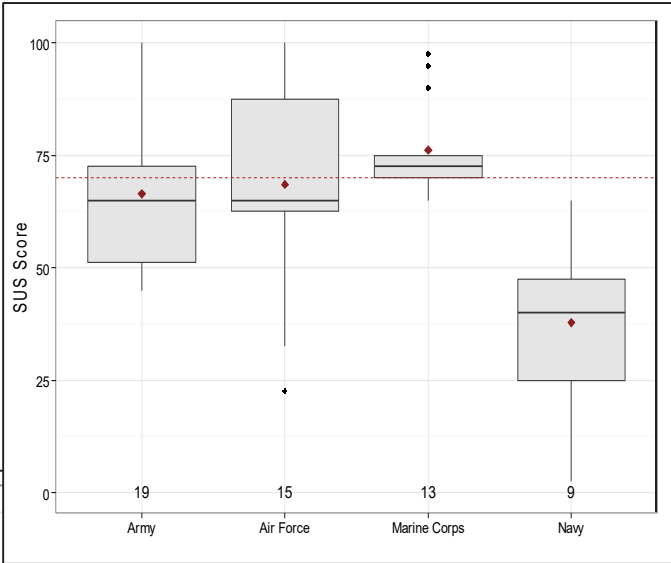
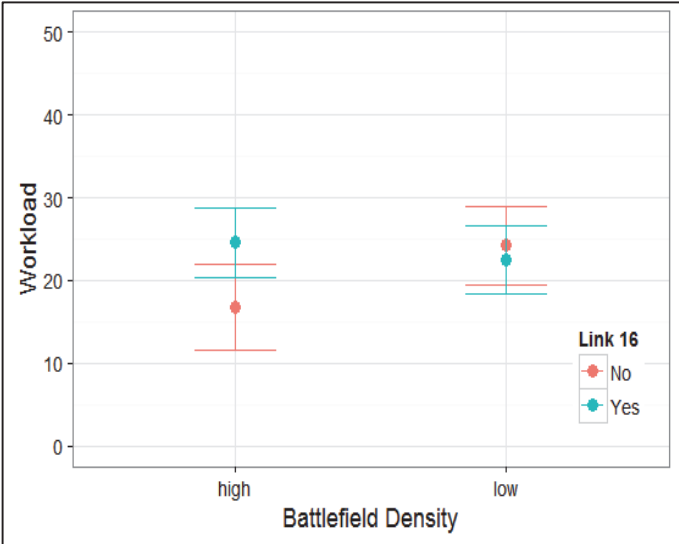
How users will be spread across test conditions

CRITICAL for evaluating the system performance AND human factors

People shape how systems perform



Operational conditions, user type, and system characteristics can all affect HSI!



Conclusion

Operational testing must consider system performance in its intended **environment** with actual **users**

Appropriate **DOE techniques** are essential to both finding software bugs and characterizing variability in high level outcomes

Capture **human-system interaction** data by collecting behavioral metrics or administering surveys using the same DOE principles