Streamlining experimental processes using bespoke software

Daniel Fay, Neville A Stanton & Aaron PJ Roberts

University of Southampton, UK

ABSTRACT

The Command Teamwork Experimental Test-bed project was a programme of work that evaluated future ways of working in control rooms through a series of command team studies. Each study required a large software stack to record the necessary data. For each study, teams of participants operated stations that required configuration before each scenario. This configuration was performed by giving spoken instructions to participants. While these instructions worked, they had issues that negatively affected the experimental procedure. To alleviate these issues, software was developed to configure the software stack for each participant. This software, the ComTET Laboratory Automation and Management Shell, has yielded benefits to both experimenters and participants. It is put forth that similar results could be achieved in other experiments by practitioners using software to bolster their experimental procedure.

KEYWORDS

Experimental procedure, automation, bespoke software

Introduction

The Command Teamwork Experimental Test-bed (ComTET) project aimed to evaluate future ways of working in control rooms using systematic and repeatable experiments (Roberts et al., 2015). A series of studies were conducted that required a large software stack for running the simulation, collecting participant data, and managing performance. A software stack is a collection of software that achieves a specified goal. Each study recruited ten teams of eight participants for high statistical power (Roberts et al., 2017; Stanton et al., 2017). One team was a professional team, and the other nine were civilian teams. Initial studies performed in the ComTET laboratory relied on participants following instructions provided by experimenters to configure their stations for scenarios. Including the experimenter commanding the team, there were nine stations to configure the correct recording and storage of data. The software stack and experimental procedure were effective, however, pilot studies revealed experimenter and time capacities had been reached. This was due to the vast volume of data being collected, large participant numbers, and a large software stack. Each study recorded participants' voice interactions, web camera video feeds, screens, and keyboard interactions. This generated a large volume of data that required stringent organisation to facilitate effective analysis. The varied nature of collected data required several pieces of software to be used, adding extra requirements and complexity to the software stack. Therefore, it was decided to streamline the experimental procedure using bespoke laboratory management software, to mitigate issues that were being encountered.

The first issue was ensuring that participants followed instructions properly, and in synchronisation with each other. Synchronisation was important as it ensured reliability and validity of results. For example, if two recordings were out of synchronisation it might appear that communication happened faster than it did. This would lead to erroneous conclusions and undermine the scientific

process. Participants had a range of computing abilities, with some requiring additional support to configure their software, or unintentionally making mistakes. Thus, additional time was required to support these participants, and to rectify mistakes. Furthermore, there would be natural variations between each participant when performing synchronised actions. These inconsistencies introduced additional pre-processing requirements, requiring additional time and resources to complete the analysis. The second issue was data being misnamed or saved in different locations by participants. Significant time and effort was expended by experimenters to organise data that had been unintentionally saved incorrectly. The final issue was the temporal demands of configuring several different pieces of software on several computers at once, using participants to do so. Even with three experimenters, a significant time penalty was encountered when configuring scenarios. Combined, these issues inflated the amount of time taken to perform experiments, introduced errors, and increased workload.

To remedy these issues, the ComTET Laboratory Automation and Management Shell (Clamshell) software was created. Clamshell was designed to control all experimental software according to experimental requirements, considering current capacity, and future requirements. It was envisaged that an experimenter could use this software to setup an experiment, creating a better participation experience. Commercial off the Shelf (COTS) software was considered, but it was either unsuitable or too expensive. The ComTET laboratory was designed to be cost efficient so purchasing software that exceeded the cost of the laboratory would not have fitted with this ethos. Additionally, software available to purchase did not have the correct functionality or would not have successfully controlled the software stack. Therefore, it was decided to utilise in-house expertise to create bespoke software. The software was to be flexible, to allow for future needs, and easily operable by any member of the experimental team.

Clamshell creation

A full requirements assessment was completed, incorporating input from domain matter Subject Matter Experts (SMEs) for fidelity requirements, and human factors practitioners for optimisation of experimental design. All SMEs had experience in the defence domain. Input was given at steering committee meetings, where experimental design was decided, and in consultations with individual SMEs when designing the ComTET laboratory. Next, an assessment of interfacing capabilities for each software program was performed. Results ranged from full source code access to limited capability for programmatic interaction. With full source code access it was possible to implement any behaviour required, whereas more limited programs could only be interacted with in a specific manner, restricting capability. For example, it was possible to specify the behaviour of the video recording software, but only simulated human input could be used with the voice recording software. When this interaction was happening, other tasks could not be started, slightly extending setup times. This wide range of capability reiterated the need for flexibility, and a variety of integration methods. Software with sufficient capability would be interacted with directly, either in code, or via command line parameters. For limited programs, software to replicate human interaction would be employed. A server application with a Graphical User Interface (GUI) was designed, to display the laboratories status, and to send commands to participant computers. Each participant computer had a client application that received and enacted these commands.

An agile (Bourque & Fairley, 2014) approach was used to create Clamshell in Microsoft Visual Studio. C# and the Windows Presentation Foundation (WPF) were chosen due to familiarity and extensive language capabilities, backed by the .NET Framework. As C# is a high-level programming language, functionality could be developed without needing to consider system specifics. This ensured development time was allocated to creating experimentation-oriented features, instead of the specifics of working with a system. Other languages offered these benefits; however, a lack of familiarity could have increased development time. During development, an

opportunity to create digitised versions of the NASA-TLX (Hart & Staveland, 1988) and Bedford Workload Scale (Roscoe & Ellis, 1990) was realised. This was to reduce experimentation time, eliminate costs of data pre-processing, and reduce paper usage within ComTET. Each test was fully integrated, and saved results in a specified location, using an understandable format. The extensibility of C#/WPF provided capability to create a framework for inclusion of similar tests in the future. A series of functionality milestones were planned, specifying which components of Clamshell would be complete, and when. Example milestones included networked communication being implemented, or integration with a new program. At these milestones, Clamshell was installed in ComTET for in-situ testing, ensuring full compatibility when deployed. The final product successfully integrated all planned software and allowed an experimenter to set up an experiment for participants. Clamshell was not static and was regularly updated between manipulations to ensure the best experimental process possible. This allowed functionality to be added for new experiments, and data collection to be refined to achieve high quality results. The core experimental procedure was not altered, rather data collection mechanisms or outputs. This avoided creating confounding variables, which could have affected results. For example, the last major upgrade included a new screen and webcam recording software package, improving video quality and recording simplicity. The increased video quality and clearer audio allowed voice communications to be better understood when transcribing. This additional quality also provided better recording resilience, as transcriptions could be crosschecked from multiple sources. Future work for Clamshell will involve updates to meet new requirements. While these updates may only be small tweaks or additions, they will ensure that Clamshell supports the needs of the ComTET project, and its experimenters, throughout its lifetime.

Effect of Clamshell

The experimental procedure was updated to include Clamshell for future experiments. Automation largely removed participant error, and ensured all actions were performed in synchronisation. An experimenter sent actions to each client, and the action performed consistently and automatically on selected participant computers. Participants were only asked to perform simple actions, such as accepting prompts, or typing in answers, which eliminated data entry issues when naming or storing data. This increased data integrity, facilitating analysis by reducing pre-processing stages such as finding missing data, or creating a consistent file structure. Previously setting up an experiment could have taken up to thirty minutes, however with Clamshell, setting up an experiment can take as little as five minutes.

Data created using Clamshell was stored in a structured fashion locally and can be collated from all participants into a central store. Previously, data had been stored in a separate location by each program, which had to be manually changed by participants, and named accordingly. Accounting for finding misplaced data, collecting and collating data from experiments using this approach took a week, or longer. With Clamshell, locations and names are pre-set, reducing misnamed or misplaced data. It is also unlikely that data will be both misplaced and misnamed, enabling absent data to be found quickly. Combined, these benefits have reduced data collection time to a few hours, a significant saving of person-power.

In summary, Clamshell has vastly increased the efficiency of ComTETs' testing protocol. This has resulted in substantial temporal savings for experimenters and participants. It has facilitated collection of large volumes of data, which positively affected outcomes from the ComTET programme. Substantial economic savings have been realised from increased efficiency and capacity, especially in comparison to commercial products affording similar functionality, with less flexibility.

Application to human factors

Clamshell has demonstrated that implementing software to support experimental procedure can have a beneficial impact for practitioners and participants. By facilitating the experimental procedure, practitioners can concentrate on running experiments, instead of handling the minutia. Participants benefit from a better experience, and maximisation of their time. While Clamshell encompassed all ComTET laboratory procedures, a practitioner could support just one aspect of their experimental procedure. In most cases, this does not require an extensive knowledge of programming, rather an understanding of the experiment, the right tools, and sufficient documentation. Finally, the concept of Clamshell is not restricted to a particular domain; it is hoped that this work leads practitioners to apply human factors expertise to their experimental procedures and reap the results of doing so.

References

BOURQUE, P. & FAIRLEY, R. E. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society

HART, S. G. & STAVELAND, L. E. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Advances in psychology,* 52**,** 139-183

ROBERTS, A., STANTON, N. & FAY, D. 2015. The Command Team Experimental Test-bed Stage 1: Design and Build of a Submarine Command Room Simulator. *Procedia Manufacturing***,** 2800-2807

ROBERTS, A., STANTON, N. A. & FAY, D. 2017. The Command Team Experimental Test-Bed Phase Two: Assessing Cognitive Load and Situation Awareness in a Submarine Control Room. *In:* STANTON, N. A., LANDRY, S., DI BUCCHIANICO, G. & VALLICELLI, A. (eds.) *Advances in Human Aspects of Transportation: Proceedings of the AHFE 2016 International Conference on Human Factors in Transportation, July 27-31, 2016, Walt Disney World®, Florida, USA.* Cham: Springer International Publishing

ROSCOE, A. H. & ELLIS, G. A. 1990. A subjective rating scale for assessing pilot workload in flight: A decade of practical use. Royal Aerospace Establishment Farnborough (United Kingdom)

STANTON, N. A., ROBERTS, A. P. J. & FAY, D. T. 2017. Up periscope: understanding submarine command and control teamwork during a simulated return to periscope depth. *Cognition, Technology & Work,* 19**,** 399-417