Automated Deduction and Usability Reasoning

José Creissac Campos
Universidade do Minho, Braga, Portugal

Michael D. Harrison
University of Newcastle upon Tyne, UK

INTRODUCTION

Building systems that are correct by design has always been a major challenge of software development. Typical software development approaches (and in particular interactive systems development approaches) are based around the notion of prototyping and testing. However, except for simple systems, testing cannot guarantee absence of errors, and, in the case of interactive systems, testing with real users can become extremely resource intensive and time-consuming. Additionally, when a system reaches a prototype stage that is amenable to testing, many design decisions have already been made and committed to. In fact, in an industrial setting, user testing can become useless if it is done when time or money is no longer available to substantially change the design.

To address these issues, a number of discount techniques for usability evaluation of early designs were proposed. Two examples are heuristic evaluation, and cognitive walkthroughs. Although their effectiveness has been subject of debate, reports show that they are being used in practice. These are largely informal approaches that do not scale well as the complexity of the systems (or the complexity of the interaction between system and users) increases. In recent years, researchers have started investigating the applicability of automated reasoning techniques and tools to the analysis of interactive systems models. The hope being that these tools will enable more thorough analysis of the designs.

The challenge faced is how to fold human factors' issues into a formal setting as that created by the use of such tools. This article reviews some of the work in this area and presents some directions for future work.

BACKGROUND

As stated earlier, discount usability analysis methods have been proposed as a means to achieve some degree of confidence in the design of a system from as early as possible in development. Nielsen and Molich (1990) proposed a usability inspection method based on the assumption that there are a number of general characteristics that all usable systems should exhibit. The method (heuristic evaluation) involves systematic inspection of the design by means of guidelines for good practice. Applying heuristic evaluation involves setting up a team of evaluators to analyze the design of the user interface. Once all evaluators have performed their analysis, results are aggregated thus providing a more comprehensive analysis of the design. To guide analysis, a set of design heuristics is used based on general purpose design guidelines. Over the years, different sets of heuristics have been proposed for different types of systems. The set proposed by Nielsen (1993) comprises nine heuristics: *simple and natural dialog*; *speak the user's language*; *minimize user memory load*; *be consistent*; *provide feedback*; *provide clearly-marked exits*; *provide short cuts*; *good error messages*; and *prevent errors*.

Usability inspection provides little indication of how the analyst should check whether the system satisfies a guideline. Cognitive walkthrough (Lewis, Polson, Wharton, & Rieman, 1990) is one technique that provides better guidance to the analyst. Its aim is to analyze how well the interface will guide the user in performing tasks. User tasks must first be identified, and a model of the interface must be built that covers all possible courses of action the user might take. Analysis of how a user would execute the task is performed by asking three questions at each stage of the interaction: *Will the correct action be made sufficiently evident to users?*;

Will users connect the correct action's description with what they are trying to achieve?; and *Will users interpret the system's response to the chosen action correctly?* Problems are identified whenever there is a "no" answer to one of these questions.

Informal analytic approaches such as those described pose problems for engineers of complex interactive systems. For complex devices, heuristics such as "prevent errors" can become too difficult to apply and validate. Cognitive walkthroughs provide more structure but will become extremely resource intensive as systems increase in complexity and the set of possible user actions grows.

To address these issues, researchers have started looking into the application of automated reasoning techniques to models of interactive systems. These techniques are generally more limited in their application. This happens both because of the cost of producing detailed initial models and because each tool performs a specific type of reasoning only. Nevertheless, they have the potential advantage that they can provide a precise description that can be used as a basis for systematic mechanical analysis in a way that would not otherwise be possible.

Automated theorem proving is a deductive approach to the verification of systems. Available theorem provers range from fully interactive tools to provers that, given a proof, check if the proof is correct with no further interaction from the user. While some systems provide only a basic set of methods for manipulating the logic, giving the user full control over the proof strategy, others include complex tactics and strategies, meaning the user might not know exactly what has been done in each step. Due to this *mechanical* nature, we can trust a proof done in a theorem prover to be correct, as opposed to the recognized error prone manual process. While this is an advantage, it also means that doing a proof in a theorem prover can be more difficult, as *every little bit* must be proved.

Model checking was proposed as an alternative to the use of theorem provers in concurrent program verification (Clarke, Emerson, & Sistla, 1986). The basic premise of model checking was that a finite state machine specification of a system can be subject to exhaustive analysis of its entire state space to determine what properties hold of the system's behavior. By using an algorithm to perform exhaustive state space analysis, the analysis becomes fully automated. A main drawback of model checking has to do with the size of the finite state machine needed to specify a given system: useful specifications may generate state spaces so large that it becomes impractical to analyze the entire state space. The use of symbolic model checking somewhat diminishes this problem. Avoiding the explicit representation of states and exploiting state space structural regularity enable the analysis of state spaces that might be as big as 10^{20} states (Burch, Clarke, & McMillan, 1990). The technique has been very successful in the analysis of hardware and communication protocols designs. In recent years, its applicability to software in general has also become a subject of interest.

AUTOMATED REASONING FOR USABILITY EVALUATION

Ensuring the quality (usability) of interactive systems' designs is a particularly difficult task. This is mainly due to the need to consider the human side of the interaction process. As the complexity of the interaction between users and devices increases, so does the need to guarantee the quality of such interaction. This has led researchers to investigate the applicability of automated reasoning tools to interactive systems development.

In 1995, Abowd, Wang, and Monk (1995) showed how models of interactive systems could be translated into SMV (Symbolic Model Verifier) models for verification. SMV (McMillan, 1993) is a symbolic model checker, at the time being developed at Carnegie Mellon University, USA (CMU). They specified the user interface in a propositional production systems style using the action simulator tool (Curry & Monk, 1995). The specification was then analyzed in SMV using computational tree logic (CTL) formulae. The authors proposed a number of templates for the verification of usability related properties. The questions that are proposed are of the type: "*Can a rule somehow be enabled?*"; "*Is it true that the dialogue is deadlock free?*"; or "*Can the user find a way to accomplish a task from initialization?*".

The modeling approach was quite naive and enabled the expression of models at a very high level