Safer “5-key” number entry user interfaces using Differential Formal Analysis

Abigail Cauchi
Future Interaction Technology Lab
Swansea University
csabi@swansea.ac.uk

Andy Gimblett
Future Interaction Technology Lab
Swansea University
a.m.gimblett@swansea.ac.uk

Harold Thimbleby
Future Interaction Technology Lab
Swansea University
harold@thimbleby.net

Paul Curzon
School of Electronic Engineering and Computer Science
Queen Mary University of London
paul.curzon@eecs.qmul.ac.uk

Paolo Masci
School of Electronic Engineering and Computer Science
Queen Mary University of London
paolo.masci@eecs.qmul.ac.uk

Differential formal analysis is a new user interface analytic evaluation method based on stochastic user simulation. The method is particularly valuable for evaluating safety critical user interfaces, which often have subtle programming issues. The approach starts with the identification of operational design features that define the design space to be explored. Two or more analysts are required to analyse all combinations of design features by simulating keystroke sequences containing keying slip errors. Each simulation produces numerical values that rank the design combinations on the basis of their sensitivity to keying slip errors. A systematic discussion of the simulation results is performed for assessing the causes of any discrepancy, either in numerical values or rankings. The process is iterated until outcomes are agreed upon. In short, the approach combines rigorous simulation of user slip errors with diversity in modelling and analysis methods.

Although the method can be applied to other types of user interface, it is demonstrated through a case study of 5-key number entry systems, which are a common safety critical user interface style found in many medical infusion pumps and elsewhere. The results uncover critical design issues, and are an important contribution of this paper since the results provide device manufacturers guidelines to update their device firmware to make their devices safer.

Number entry, stochastic simulation, medical devices, interactive systems, blocking errors.

1. INTRODUCTION

Best practice for designing effective and safe interactive systems uses methodologies that were developed primarily in office and consumer domains: iterative design, user evaluation (using both laboratory and field experiments), and so forth; international standards, e.g., ISO 9241, summarise current best practice. However, safety critical and dependable applications should be designed not just to be usable, but to be safe; design should reduce risk to be As Low As Reasonably Practical, ALARP, which is a legal requirement under the UK Health & Safety At Work Act (1974) and under similar legislation in other countries.

Dependable interactive applications, we argue in this paper, require different methodologies than conventional usability approaches. For example, a standard laboratory experiment may find that users prefer one system to another, or that they make fewer errors or are faster. This is certainly useful information, but (except for very simple systems) a lab study cannot cover all features (let alone all states and transitions) of a system. If the interaction design has bugs — actual software bugs or poor boundary cases in the user interface — then human participant-based evaluation may not help enough. For complex systems, and for critical applications, reliance on user testing alone may not be good enough to assure a system has as few design defects as possible.

A common approach to assessing human factors is via empirical studies. With any method, its validity is an important issue. In a typical usability experiment researchers try to achieve validity by managing participant variability. For example, if the only participant was a university student, the results would not be representative of a typical consumer population; in general the smaller (and less representative) the population of participants the less reliable it is to estimate the significance of any results. In addition, running large trials is prohibitively expensive.

Proceedings of the BCS HCI 2012
People & Computers XXVI, Birmingham, UK

In our approach, we replace variability among participants with variability among implementations: any detectable variability has an explicit traceable cause, and any lack of variability is a consequence of precise requirements. In our analyses we used stochastic simulation, a very fast way of generating data; comparably diverse empirical experiments using human participants would have taken excessively long. These differences between conventional approaches to usability evaluation and our approach create new opportunities, particularly relevant to dependable design.

1.1. Case study and its importance

For a case study we use numerical formal methods, in particular stochastic methods. We will use "out by ten error" (defined below) as a measure of error (or dependability), particularly relevant to the chosen application of number entry.

As a concrete example, this paper considers the task of interactive number entry. Incorrect drug doses and incorrect drug dose calculations are a significant contributory factor in unnecessary fatalities in healthcare. There are many papers on the prevalence of prescribing errors (e.g., Dean *et al*. 2002), but very few on user interaction errors, since interaction errors are harder to measure as they generally do not leave a paper record that can be easily analysed. Vicente *et al*. (2003) estimate the probability of fatal number-entry errors on PCA pumps (ones controlling pain, typically delivering opiates) as between 1 in 33,000 to 1 in 338,800 (the large uncertainty is due to estimating reporting rates — many errors are not reported); or in absolute terms approximately 65–667 per year in the US or (scaling by population) 155–1587 per year in Europe. Vicente *et al.* warn that these are low estimates as they are based on fatalities in the US but the PCA pump is used worldwide, and hence the denominator used, the number of pumps sold, would have been too high.) By way of comparison, the probability of death from general anæsthesia is approximately 1 in 200,000–300,000.

This work is motivated by the vast interaction differences between implementations of number entry systems in popular, commercial medical infusion pumps. The 5-key number entry system (see figure 1) is gaining popularity in infusion pumps from leading manufacturers such as BBraun and Zimed and by studying these pumps we found that the same keying sequences in apparently identical number entry interfaces result in very different outcomes.

Consider the case where the starting screen displays `0` and our goal is to input a dose of 950 mL; on one commercial pump the key sequence ◀ ▲ ▲ ▲ ▲ ◀ ▼ results in a display of `950` — but keying in the same sequence starting from the same state on a different commercial pump results in `000.1`. A detailed and formal description of why this happens may be found in Masci *et al*. (2011). We recognise that different interaction design choices lead to different values on the display (see figure 2 for some examples, and section 3); we are therefore concerned about finding the best combination of choices (along with their rigorous specifications) to make the design more resilient to human error.

Figure 1: *An example of 5-key user interface layout. Here the cursor is shown in the left-most position, and the display format is suitable for entering times, 0 minutes to 999:59 hours. Some 5-key interfaces omit the OK button as its use can be implied by the user performing any action with any non-arrow button.*

Design choice	Press	Display
Arithmetic	▲	09 → 10
Independent dial	▲	09 → 00
Wrap	▶	09 → 09
No wrap	▶	09 → 09
Left start		09
Right start		09
Block underflow	▼	00 → 00
Underflow & arithmetic	▼	00 → 99
Underflow & independent dial	▼	00 → 09

Figure 2: *Examples of design choices. Note how underflow/blocking and dial/arithmetic interact. Different interpretations of these and other feature interactions affect the sensitivity of the user interface to user error.*

In differential formal analysis, a formal analysis is performed multiple times in parallel by a group of investigators, using a variety of implementations, preferably across a variety of implementation platforms. Differences between platforms, the implementation techniques they admit, and individual team members' understanding of the task at hand all lead naturally to differences between implementations and results obtained. These differences provide points of discussion and further investigation, highlighting ambiguities and areas of underspecification in the problem description against which the team is working. The aim is not necessarily to eliminate all such differences — leading to "one true" well-specified problem description, implementation and set of results (whose cost may outweigh its benefit) — but

rather to expose issues which might otherwise easily be missed, to deepen understanding of the situation, and to direct further investigation.

Differential formal analysis raises specific design issues that would have been very easy to ignore after doing conventional usability studies. In conventional usability studies, individual differences are a strong source of experimental variability, so differences are expected and need no further explanation. In contrast, the proposed quantitative analysis, being systematic, eliminates variability due to individual differences or other sources, such as experimenter effects, and thus makes it much easier to spot and hence consider the impact of design features. More precisely, if the proposed analysis uncovers variability, this is due to misunderstandings in requirements that can then be uncovered and resolved — it uncovers specific design issues rather than generic psychological issues that, even when statistically significant, design cannot readily address.

1.2. Related work

KLM and GOMS and their variants (Card, Moran & Newell 1983, etc) are well-established evaluation methods that are useful for obtaining a measure of time to perform a specified goal. These techniques generally assume no user errors, and evaluate unit tasks (CogTool is a recent tool that partly automates this process). In contrast, in our approach, we are specifically concerned with user error and how to design to manage it better, and we do not consider tasks like "enter a number" as a unit task — the user may make errors within the task that need to be carefully analysed. We are not so much interested in time as in error rates; if we can make a user safer that is more important than making them faster. More accurately, finding out how to make them safer is more important than making them faster: we want to compare designs on safety. Conventional evaluation methods do not help here. Furthermore, as this paper makes clear, not only must we consider user error, but also designer/analyst interpretations of "enter a number" — it turns out to be a much richer and more complex design problem than expected; in a sense, we have to account not just for user error but designer error as well!

In the design of safety critical number entry systems, the "best" design is not necessarily the fastest or most appealing to users. In safety critical domains, having a design that reduces errors is desirable, however, we highlight that design is a trade-off — in general, we aim at achieving an appropriate balance between speed and safety. In the KLM-GOMS paradigm approaches, errors are not taken into consideration and it is assumed that users do not make errors. On the other hand, in our approach, human error (both user and designer) has an integral role, making the approach suitable for helping evaluate safety critical designs.

Oladimeji, Thimbleby & Cox (2011) empirically compare so-called serial and incremental number entry interface styles. They used eye tracking, and uncovered important design principles (including an explanation of why incremental interfaces are more dependable than numeric keypad interfaces for number entry). Our current work compares 5 different design features in up to 28 combinations: this scale of comparison complements Oladimeji *et al.*'s empirical work by targeting subtle variations in interface layouts which previously were all classed under a single "incremental" heading.

Fields (2001) explores the consequences of different kinds of error being made, based on a similar classification to ours. He developed a finite state transition notation for describing task models that can be combined with a device model. Combined models were then analysed using off-the-shelf model checking technology to analyse the effect of executing tasks on the device. He also defined patterns of user error that could be introduced into the model based on a similar classification to ours (e.g., omission or repetition of action). The consequences of the introduced errors could then be investigated via model checking. Fields considers exploring underlying cognitive causes, an approach further considered by, for example, Curzon, Rukšėnas & Blandford (2007). Our work here offers a different solution — to consider sensitivity analysis.

This work complements Masci *et al.* (2011), which defined predictability in higher order logic, and explored how such a property can be verified on real systems through automated reasoning tools. The predictability property tests whether an expert user can tell what state the device is in from the perceptible output of the system, and hence accurately predict the consequences of an action from that state — normal human users can do no better. The analysis was performed on the formalisation of two real devices, and showed that devices, when closely examined, have many boundary cases where interactive functionality seems awkward. Here we explore the impact of errors, and assess in a systematic way if variations in the design of the numeric entry system can reduce harm when errors are made.

Following Thimbleby (2006), we performed our work retrospectively by initially reverse-engineering commercial products. Had we worked in the development teams, we could have proceeded exactly as described here, except we would

have been implementing devices directly from requirements or perhaps from prototypes, rather than by reverse engineering. An important aspect of our research is that it is deliberately handling design issues on a realistic scale, with realistic quirks and issues. The faithful reverse engineering ensures our systems are the same complexity as commercial product features. In other words, our methodology can be applied to commercial development of dependable interactive products. In fact, as anticipated by Thimbleby (2006), the discipline of reverse engineering itself uncovered numerous design questions that are important to consider in safety critical systems.

2. DIFFERENTIAL FORMAL ANALYSIS

The Differential Formal Analysis (DFA) process is illustrated in figure 3; it starts by determining optional design features which are either implemented or not. A design is a combination of features, and all the combinations make up the design space. Two or more researchers then use Stochastic Key Slip Simulation (SKSS) detailed in section 2.1, to rank the designs.

SKSS entails generating a large number of key sequences that take us from one number to another and inserting a keying error (substitution, deletion, repetition, or transposition) with probability p per keystroke. If the actual and intended result values differ by more than a certain magnitude k (say, 10), we count the error; we then rank according to the proportion of "out by k" errors.

SKSS reveals further design and evaluation issues when performed by independent researchers. In our case study we found that the features were not initially described formally enough and numerical disagreements raised clear questions about how people enter numbers which we would not have noticed otherwise. After discussing disagreements, then (if any), the features are refined, SKSS implementations are modified to reflect this, and ranks are determined again. This iteration happens as many times as necessary for the researchers to converge on results.

The DFA process thus enforces rigorous science by ensuring that results are repeatable (4 researchers independently repeated the case study analysis), and thoroughly discussed. In effect, the important safety critical issues are well thought out and we can have more confidence in our results.

2.1. Stochastic key slip simulation (SKSS)

Here we introduce the analytical technique at the core of the DFA process: a stochastic simulation where *slips* are introduced into sequences of key presses in interactive number entry systems, in order to explore the trade-offs arising from various choices in the design of such systems, as previously developed in Thimbleby & Cairns (2010) for numeric keypad user interfaces.

The basic approach here simulates a human changing the display from one value to another, but allowing for — and analysing the sensitivity of the design to — human error. The user will make keying slips: repetition (or key bounce); transposition (switching two keys in the sequence); deletion (accidentally not pressing a key, or the device not registering a key press); substitution (pressing a different key than intended). These slips are modelled with some probability p per keystroke (which of course may depend on environmental factors), and the proposed designs need to be evaluated for the consequences of those slips. Some design features will make a design more sensitive to such slips: the more sensitive a design, the more likely it is that slips will lead to uncorrected unintended consequences; such sensitivity is best avoided.

It is routine in several safety-critical domains, such as healthcare, to consider "out by ten errors," where the number used is a factor of ten out from the number intended. This suggests a clear measure of design error sensitivity: run simulations to determine the dependency of out by ten errors on p.

2.2. Setters, solvers and slip models

A simulation is implemented in terms of **solvers**, **setters**, and **slip models**. Solvers and setters occur in matched pairs for each design. A solver generates user key sequences that solve the task in question; a slip model inserts slips into key sequences; a setter executes key sequences.

For number entry analysis, we take the task to be to change the display from showing some number m to displaying n and entering it to the underlying application (hence, the fifth key, OK). Thus, each design's **solver** is a function that takes two numbers m, n and generates a sequence of key presses to change the display from showing m to show n within the constraints of the chosen design, and then submits it by pressing OK. In the special case $m = n$, the solver just generates OK.

Different approaches to user modelling are possible: the simplest (conceptually, if not in terms of implementation) is to compute the optimal sequence for the given task. In reality, users tend to find *satisfactory* rather than optimal solutions, and some truly optimal sequences may be cognitively too hard to determine (Simon 1996). Therefore, the approach as used in this work finds the

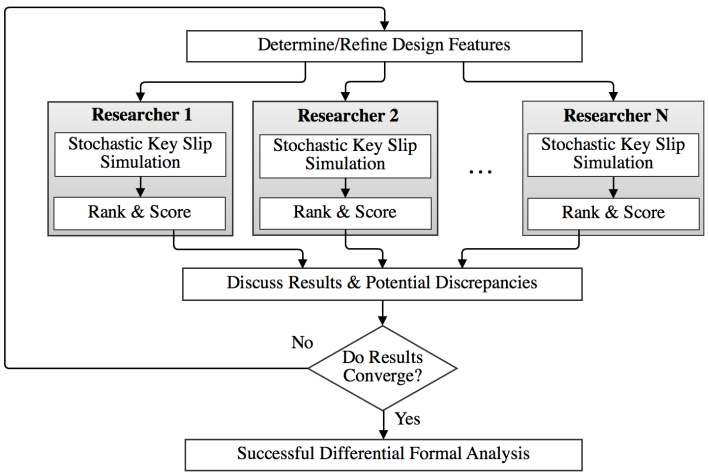

Figure 3: *The Differential Formal Analysis Process*

sequence with either monotonic left-to-right or right-to-left cursor motion, a realistic satisficing strategy. Complementary empirical studies would give us a better insight into how users enter numbers, however our satisficing solver (which the authors agreed on) is sufficient for the purpose of demonstrating the process.

In contrast, each design's **setter** takes an initial value m and a sequence of keystrokes σ, and returns a triple, $r = \langle r_v, r_n, r_e \rangle$, the result of applying σ to the system from the starting point m. r_v is the actual final value reached, r_n is the intended/target value, and r_e is true if an error occurred and was blocked — see section 3 for details of blocking as a design choice. Some setters do not block errors under any circumstances (e.g., this is how some 5-key designs work). No blocked error does not imply $r_v = n$, since $\neg r_e$ means no error was *blocked* by the setter, not that no error occurred — for example, slips could occur such that a different number was entered 'correctly' without triggering a blocked error.

A **slip model** generates a sequence of keystrokes with keying slips; specifically, given a key sequence σ and a probability p, $\mathit{slips}(\sigma, p)$ inserts slips at each keystroke of σ, each with independent probability p. The slip model is the way the analysis models human error. Then, for some device design d,

$$\mathit{set}_d(m, \mathit{slips}(\mathit{solve}_d(m, n), p))$$

"tries" to set the display to n given that it initially shows m. As p increases, this becomes increasingly unlikely. By using different setters, the idea is to find out which combinations of design features make the setter robust against errors introduced as a consequence of the slips the slip model introduces.

The keystroke slip model simulates human keying slips. As described above, in our experiments we explored uniform distributions of several forms of keystroke slip, namely: substitution, repetition, omission, and transposition. Wiseman, Cairns & Cox (2011) present a taxonomy of number entry errors. The slip model used here is able to take into account all error types described in that taxonomy, and it could be generalised to arbitrary error distributions.

2.3. Error sensitivity

Given a design d, a set of probabilities P, and a set T of task pairs (m, n), an **experiment** calculates for each $p \in P$ the set $S_{dT}(p)$ of results of running the appropriate solver, slip model and setter on the various (m, n) pairs:

$$S_{dT}(p) = \{\mathit{set}_d(m, \mathit{slips}(\mathit{solve}_d(m, n), p)) \colon (m, n) \in T\}$$

The **error sensitivity** $e_d(p)$ for a design d at p is then:

$$\frac{|\{r \in S_{dT}(p) \colon \neg r_e \wedge (r_v \geq k r_n \vee r_v \leq r_n / k)\}|}{|\{r \in S_{dT}(p) \colon \neg r_e\}|}$$

We take $k = 10$. That is, from each set of samples, we count the number of non-blocking paths which resulted in an out-by-ten error, and divide it by the total number of non-blocking paths. Note that we also require *valid* random values of m and n (which may depend on the application).

The error sensitivity of a design may then be investigated, and compared with other designs. A better design is less sensitive to error, but since the error sensitivity depends (as defined) on p, a simpler measure is the mean gradient $de_d(p)/dp$ around typical p values (e.g., $p \approx 0.001$); in fact, for our case studies, for typical p, sensitivity is nearly linear, and hence effectively independent of p. A lower gradient is better.

Because of linearity, the best design decisions do not depend on the actual value of p; it is not necessary to

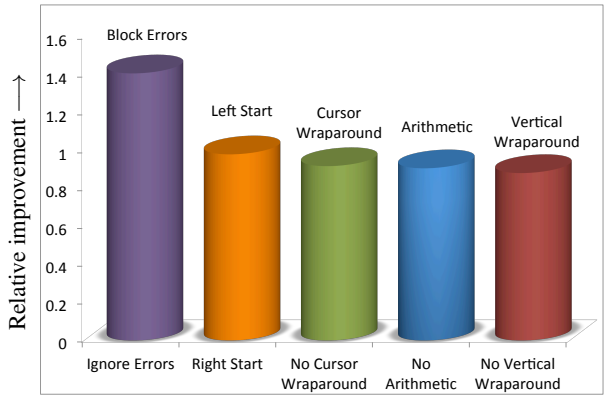

Figure 4: *Summary of C# analysis findings. Features mentioned at the top of the bars are better, by the factor shown. Specifically, over all design choices, blocking errors reduces the mean sensitivity of the 5-key interface by a factor of 1.41, start at left is better than start at right by a factor of 0.98, cursor wraparound is better than no cursor wraparound by a factor of 0.91, arithmetic is better than no arithmetic by a factor of 0.9 and vertical wraparound gives us an improvement over no vertical wraparound by a factor of 0.88.*

perform empirical experiments to determine p. In an important sense the resulting design is more robust — as it does not depend on specific assumptions of user performance (as measured by p). Similarly, if we establish that the results are broadly independent of the mix of slip types (transposition, deletion, etc) then there is no need to model the user more realistically!

3. 5-KEY NUMBER ENTRY DESIGN

As a case study we applied DFA on 5-key number entry systems to find the system which is least sensitive to human error. This type of interface (shown in figure 1) is popular in commercial medical infusion pumps and it is safety critical since it is the interface used to enter drug doses. The 5-key system uses a cursor and arrow keys to change digits and an [OK] key to submit the number.

From reverse engineering infusion pumps from different manufacturers we found that there are implementation variations on how the cursor works and how the digits work. These make up our design features and are described here.

Wraparound — if applied to digits, if the digit is at the maximum and [▲] is pressed, the digit goes to the minimum value and vice versa for pressing [▼] on the minimum value. If wraparound is applied to the cursor, if [◄] is pressed on the leftmost position, the cursor goes to the rightmost position and vice versa.

Arithmetic — if a display shows 09 and [▲] is pressed, the display shows 10. Hence, when arithmetic is on, simple arithmetic operations are performed on the display through the [▲] and [▼] keys.

After implementing the four simulations independently we found disagreements on how wraparound and arithmetic interact. One understanding was that if wraparound and arithmetic are both on at the same time, then the wraparound function happens at the minimum and maximum values of the entire displayable value. The other possibility is that wraparound and arithmetic can not be present together, and one is the inverse of the other i.e., an interface either treats the digits independently (i.e., in a dial style) or does not. This shows that what seem to be clear design feature requirements are easily misinterpreted through the details and this process raises awareness to these issues which are important for safety critical systems.

Cursor start position — this is the choice of whether the cursor starts on the left or on the right when the number entry interface starts.

Block errors — some interfaces alert users when the user has done something "wrong." Of course, without knowing the intended number n, no system can do this reliably. Certain circumstances, however, might prompt a warning. For example, attempting to move a cursor beyond the bounds of its display might result in such an alert; alternatively, the user's attempt could be silently ignored, or even honoured (in some sense) by wrapping the cursor around to the other end of the display (a design choice we investigate in section 3). Many systems assume $n = 0$ is not a valid number — for example, an infusion pump is not needed if the drug dose is zero, so this is a detectable error.

A system that alerts the user to slips might be seen as less forgiving, but the payoff is (potentially) greater resilience. If the interface displays an alert, blocking further interaction until it is cleared, the user will usually become aware of the problem, and can recover from it. For example, suppose a user's task of "get from m to n" is interrupted because of a detected error, with the display in state m'; it is then reasonable to suppose that the user effectively abandons their old task, and adopts a new one, namely "get from m' to n."

3.1. Implementations

Four independent researchers (the authors Cauchi, Gimblett, Masci and Thimbleby — we advocate several analysts should be used, but our choice of four is arbitrary) implemented SKSS to analyse designs made up of all combinations of features. The probability of out by 10 error for each design

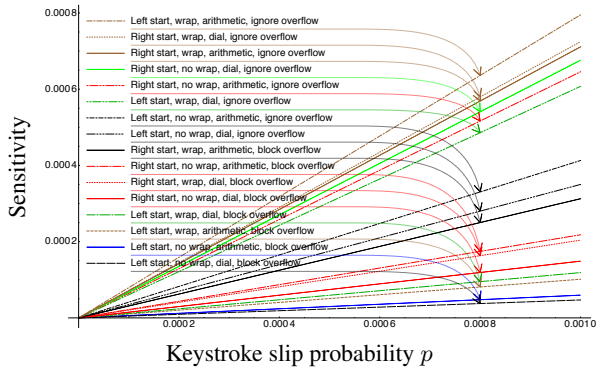

Figure 5: *Sensitivity against p for various combinations of design feature (from Mathematica). The lower gradient lines represent better (less error-sensitive) designs.*

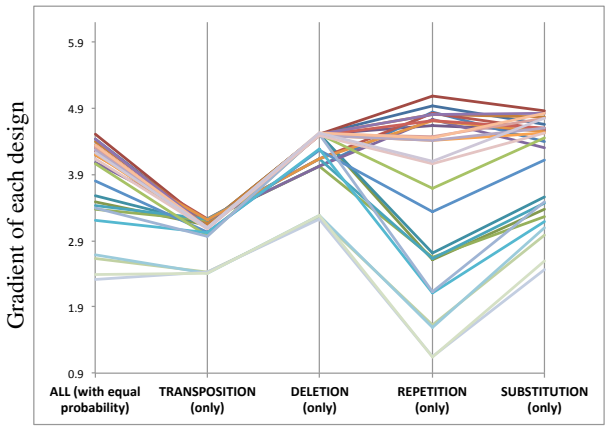

Figure 6: *Parallel Coordinates visualisation showing the relation of design rankings depending on slips*

was used to determine a rank. Each researcher carried out slightly different experiments because of different understandings of the features, and this raised important issues for discussion. The implementation platforms used were Mathematica (Wolfram 2003), C#, Möbius (Clark *et al*. 2001) and JavaScript; implementation details and variations are described in Thimbleby, *et al*. (2012).

The main issues raised from implementing SKSS were about how users key in numbers and how the features behave at the boundaries. To simulate number entry, we considered finding the best possible path from one number to another but we found that programming solvers for the best path was complex and it is unlikely that a user enters numbers in this way. We agreed to implement realistic, simple solvers which consider shortcuts users are likely to take; however these raised issues are worth studying empirically, especially since the domain is safety critical.

After a few iterations to improve and synchronise the SKSS implementations, the researchers converged on the results presented here. Figure 4 shows an example of dependence of error sensitivity on whether a feature is on or off. Block errors is the feature that most improves the design, followed by left start, cursor wraparound, arithmetic and vertical wraparound.

Figure 5 shows typical results, plotting sensitivity against keystroke slip probabilities. Because the sensitivity/keystroke error probability has an excellent linear correlation ($R^2 = 0.995$ or better) the rank order — and hence the recommended design decisions — do not depend on the value of p; in other words, doing an empirical experiment to determine p does not seem necessary.

The parallel coordinates plot in figure 6 shows the relation of how the ranking scores of each different design changes depending on what type of slips we have in SKSS. The vertical lines in the diagram represent each slip error type present in the experiment: all slips with equal probability of being chosen; transposition errors only; deletion errors only; repetition errors only; substitution errors only. Each design is represented by a polyline passing through each of the vertical lines and the position of each vertex on the a line corresponds to the score value of each design. This visualisation shows that the best designs remain best regardless of the specific mix of user errors therefore our key design recommendations do not depend on the mix of slips users make.

For most of our results we ranked our designs based on their resilience to out by 10 error. We ran an experiment to find out what happens if we consider other magnitudes of error and we can see the relation of rankings in figure 7. Here we have a parallel coordinates visualisation that shows how the probability of error (on the vertical axes) changes when considering different out by k errors. Each polyline in the diagram represents a combination of design features. The lower the polyline across the vertical axes, the smaller the sensitivity of the design to keystroke errors (and, thus, the better the design). The interesting find in this trial is that the best four designs remain toward the bottom of the visualisation and we do not have interleaving between the best designs. There are some interesting interleavings in the worse designs, however we are not interested in this since those designs with significantly higher sensitivity should not be implemented.

3.2. Special case: the BBraun Infusomat Space

We applied our analysis to an existing JavaScript simulation of the VTBI (Volume To Be Infused) number entry system of the BBraun Infusomat Space

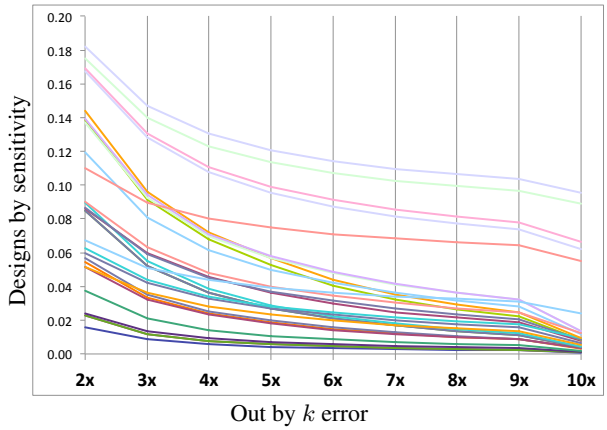

Figure 7: *Out by k errors for different design types.*

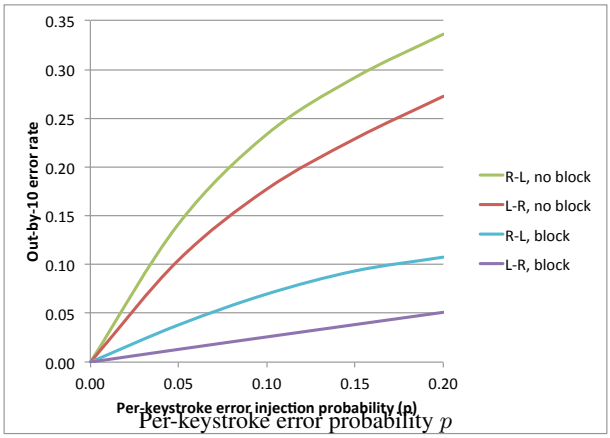

Figure 8: *Results for BBraun experiments: entry left-to-right admits less out-by-10 errors than right-to-left, and blocking errors reduces out-by-10 errors still further.*

infusion pump (also analysed in Masci *et al*. 2011). This simulation predates the analytical approach described in this paper, and was initially produced for other purposes; it is particularly interesting as its behaviour is less "regular" than the other simulations we performed. Its basic behaviour is arithmetical, without cursor wraparound; it admits entry of numbers over a range of magnitudes (see below); it has a simple memory facility around its maximum values (apparently in order to allow users to easily undo accidentally hitting the maximum value under some circumstances); and it has range-dependent non-zero minimum values, which can prevent entry of certain syntactically valid values (again, see below). The simulation thus reflects that real number entry systems frequently exhibit irregular behaviours arising from domain-specific design decisions. A set of 138 unit tests provides evidence that it is a usefully faithful simulation of the actual device behaviour: while there may be corner cases which we missed, we are confident that the simulation is faithful enough that our analysis is well-founded; in particular, the key non-regular behaviours described above are well explored by these tests.

We performed four experiments on this simulation: switching error blocking on and off, and exploring both left-to-right and right-to-left strategies in each case. The results of the analysis agree with the other analyses described above, in particular that the left-to-right strategy admits less out-by-ten errors than right-to-left, and that activating error blocking reduces the number of errors dramatically (see figure 8). Note that the actual device does beep and displays an error message under the conditions where we block errors, effectively drawing them to the user's attention. Users can adopt whichever directional strategy they prefer, though right-to-left is the natural strategy when starting from 0, as is usually the case. There are two particular aspects of the device's design which have interesting implications both for the design of the experiment, and for actual users:

First, it admits number entry over a range of magnitudes: while the display can show up to five digits, the range displayed is a sliding window between thousandths and ten-thousands; a non-zero digit in some (high-magnitude) positions can prevent access to other (low-magnitude) positions accordingly. However, the window does not slide uniformly, and there are in fact three possible ranges: *hundredths to tens* (e.g., `99.99`), *tenths to hundreds* (e.g., `999.9`), and *units to ten-thousands* (e.g., `99999`); note that the first two ranges are four digits wide, whereas the third is five wide. It is not possible to enter `9999.9`, for example. This irregularity slightly complicates random number generation, but the main complication arises in the solver. Consider the path from `99999` to `99.99`. A left-to-right strategy will have no problem here because it starts by zeroing high-magnitude columns; however, a right-to-left strategy should begin by setting the hundredths column, but the cursor won't move right past the units column while any column above hundreds is non-zero. The solution is to do what a human would do: clear columns whose content is blocking access to other required columns, before proceeding with either directional strategy as desired. (The reverse path, `99.99` to `99999`, is not problematic, however: low-magnitude columns do not block access to high-magnitude ones.)

Second, in the *hundredths to tens* range, the lowest non-zero value the device allows to be displayed is `0.1`, which is *not* the lowest value in that range (i.e., `0.01`); no value smaller than `0.1` can be displayed, in fact. Again, this irregularity complicates random test generation, and rather complicates the solver (for the right-to-left strategy, at least). Consider the path from `01.01` to `10.01`: here, the right-to-left strategy starts by zeroing the units column, which results in a jump to the minimum value `00.10`; a "blind"

subsequent adjustment of the tens column gives a final display of `10.10`, which is incorrect. In order to find a correct path, then, the solver detects such cases and simply repeats the right-to-left strategy — in this case successfully setting the hundredths and tenths columns in that order to give `10.01`. Again, we speculate that this is how a human would deal with such a situation — indeed we see no simpler way to do so.

4. DISCUSSION

Stochastic simulations have the advantage that they can be run relatively briefly to get a "quick and dirty" set of results, and then (if desired) run for longer in order to get increasingly accurate results. By comparison, methods such as model checking (Clarke 1999) tend to be (or aim to be) exhaustive by default, and they generally do not give numerical results, so comparisons between different design choices are much harder. Indeed, the concept of "user error" is a research question in itself, that needs solving before model checkers can be used effectively for differential design comparisons.

Although the analysis can tell us which design is theoretically least sensitive to human error, we cannot assume that this design is easily understandable by users. Thus, user trials retain an important role and they are essential support for results from differential formal analysis. Ultimately, design is an act of balancing competing concerns; while differential formal analysis can highlight and prioritise particular trade-offs, it is of course simply another tool, not a magic bullet.

Another potentially problematic issue for users of the technique is its cost in terms of human expertise and time. We recommend employing a team of several programmers, ideally skilled across multiple platforms in order to maximise diversity — and you need the time for them to iterate the process, including the time spent discussing their results and refining their designs and their understanding of the domain they are working within. In this respect, DFA has much in common with many formal methods!

4.1. Use in procurement

Procurement is a dual of design: you evaluate alternative designs to choose which ones to purchase, whereas in design, evaluation informs which features to implement or improve. Like our research, procurement is generally in the position of being offered finished products to evaluate, so it would be natural to reverse engineer them and apply the proposed methodology to search for critical features that may distinguish the products on offer. However, because the companies selling the devices offered for procurement would then have a financial stake in the outcome, it is likely that an extra step would be feasible: the manufacturers could either provide models for evaluation (though this may accidentally leak proprietary information), or they could help procurement check the validity of their models (this better controls access to proprietary information). The approach proposed in this paper could give procurers a way to evaluate which interface design works best towards reducing number entry error rates.

4.2. Empirical questions raised

Starting at the left is less sensitive to error than starting at the right. A possible explanation for this is that if, as a user performs a sequence of keystrokes that move from right (least significant) to left (most significant), then as errors accumulate the overall numerical effect gets larger and larger; from left to right, the opposite is true. On the other hand, perhaps in reality, starting on the far left raises the risk of confusion about which column the user is modifying — so in practice, it might turn out that right-to-left is the best strategy overall. Thus the empirical question: what do users do, and do they make slips we are not yet modelling when they start on the left or right?

We devised suitable and (we think) realistic strategies for our solvers, but we have no evidence that they accurately reflect how users find paths between numbers in 5-key interfaces. In particular, when wraparound and arithmetic are available, do users take advantage of them to reduce path length? Can they reliably find the shortest path, or is this strategy too complex to employ in reality?

We found that a feature which we call "block errors" is important to implement. A related empirical question, then, is how should "block errors" be implemented to best alert the user of a possible keying error? Do we stop interaction and ask the user to start over? Do we alert (through sounds, vibration, flashing etc.) and let the user continue entering the number? There are probably other ways of doing this: it is worth finding out and getting it right.

5. CONCLUSIONS AND FUTURE WORK

Even skilled users make slips, and so far as possible the interactive systems they use should be designed to detect and help users manage as many of their slips as possible, to help them avoid the slips turning into errors that lead to adverse situations. In the medical domain, where 5-key user interfaces are common, we wish to reduce the number of drug over and under doses. This paper has shown that even such "simple" user interfaces have a variety of subtle design choices that can be used in combination to

make a significant difference to their sensitivity to user error. In particular, we recommend that user interfaces attempt to block user error (e.g., beeping or otherwise reporting detectable errors to the user, rather than ignoring them — as is common practice). This and our other recommendations are based on a very diverse set of formal simulations, and thus are independent of the usual implicit design assumptions. These are significant results that can lead to practical applications in real, safety critical environments.

Differential formal analysis is a new methodology which should be used to complement user trials for more rigorous evaluation of safety critical number entry systems. Subtle interaction design choices in number entry lead to drastically different outcomes; it is crucial that we explore these choices and implement a design which is resilient to human error. More broadly, we found that the numerical disagreements between formal simulations led to very useful debate and insight into the details of design choices that we had previously failed to appreciate. What we now call differential formal analysis is clearly a promising design methodology to add to the toolbox.

The issues and process presented are globally important and it is critical that we, as a community, get it right. Although these are clearly human-computer interaction issues, differential formal analysis, which focusses on *safe* design, is not conventional and further work is necessary to bridge the gap between human-computer interaction and *safe* human-computer interaction.

ACKNOWLEDGMENTS

Funded by CHI+MED: Multidisciplinary Computer-Human Interaction research for the design and safe use of interactive medical devices project, www.chi-med.ac.uk, UK EPSRC Grant Number EP/G059063/1.

REFERENCES

S. K. Card, T. P. Moran and A. Newell. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–251, Washington, DC, USA, 2001. IEEE Computer Society.

P. Curzon, R. Rukšėnas, and A. Blandford. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing*, **4**(19):512–550, 2007.

B. Dean, M. Schachter, C. Vincent, and N. Barber. Prescribing errors in hospital inpatients: their incidence and clinical significance. *Quality and Safety in Health Care*, **11**(4):340–344, 2002.

R. E Fields. *Analysis of erroneous actions in the design of critical systems*. DPhil thesis, University of York, 2001.

E. M. Clarke Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Boston, NJ, USA, 1999.

P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby. On formalising interactive number entry on infusion pumps. In *FMIS2011, 4th International Workshop on Formal Methods for Interactive Systems*, 2011.

P. Oladimeji, H. Thimbleby, and A. Cox. Number entry interfaces and their effects on error detection. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction*, Volume **IV**, INTERACT'11, pages 178–185, Berlin, Heidelberg, 2011. Springer-Verlag.

H. A. Simon. *The Sciences of the Artificial*. 3rd edition, The MIT Press, Boston, MA, USA, 1996.

H. Thimbleby. Interaction walkthrough: Evaluation of safety critical interactive systems. In G. Doherty and A. Blandford, editors, *Proceedings The XIII International Workshop on Design, Specification and Verification of Interactive Systems — DSVIS 2006*, *Lecture Notes in Computer Science*, **4323**:52–66. Springer Verlag, 2007.

H. Thimbleby and P. Cairns. Reducing number entry errors: Solving a widespread, serious problem. *Journal Royal Society Interface*, **7**(51):1429–1439, 2010.

H. Thimbleby, A. Cauchi, A. Gimblett, P. Masci, and P. Curzon. Evaluating safer 5-key number entry user interface designs using differential formal analysis. Technical report, Swansea University, 2012.

K. J. Vicente, K. Kada-Bekhaled, G. Hillel, A. Cassano, and B. A. Orser. Programming errors contribute to death from patient-controlled analgesia: case report and estimate of probability. *Canadian Journal of Anesthesia*, **50**(4):328–332, 2003.

S. Wiseman, P. Cairns, and A. Cox. A taxonomy of number entry error. In *British Computer Society HCI Conference*, 187–196, 2011.

S. Wolfram. *The Mathematica Book*. Wolfram Media, 5th edition, 2003.