Avoiding latent design conditions using UI discovery tools

Harold Thimbleby
Future Interaction Technology Lab
Swansea University
Swansea, Wales, SA2 8PP
harold@thimbleby.net

To appear: *International Journal of Human-Computer Interaction* **26**(2):1–12, 2010.

ABSTRACT

Motivation – Designers make decisions that later influence how users work with the systems that they have designed. When errors occur in use, it is tempting to focus on the active errors rather than on the latent design decisions that framed the context of error, and fixing latent conditions can have a more general (and future) impact than addressing particular active failures. **Research approach** – A constructive computer science approach is used, and results from a simulation reported. **Research limitations** – Error is a complex multidisciplinary field; this paper makes a new contribution complimentary to human factors engineering. **Take away message** – This paper shows that latent design decisions cause serious problems (including fatalities) in safety critical applications; the paper proposes ***UI discovery tools*** to identify and manage latent errors. UI discovery enables human factors engineers and programmers to work together to help eliminate broad classes of latent design errors.

Keywords
UI discovery tool, active error, latent error, constraints, frame error, model checking, infusion pumps.

INTRODUCTION

Healthcare makes the consequences of decisions obvious and important. When an adverse clinical event occurs, there are formal procedures for reporting and analyzing the factors that led to the event. Many adverse events occur as a combination of human actions and use of interactive medical devices, for example a nurse presses the wrong button on a device and it then gives a patient the wrong dose of a drug. The ISMP (2004) reports that in 55% of adverse drug events the device operated as designed (the 55% would be much higher if the baseline was only interaction errors, such as button press errors, rather than including, as it does, non-device interaction errors such as wrong drug, wrong route). The implication is that the causes of adverse events are in the domain of human factors rather than in design, because if the device works as designed then the causal factors of the error must be located elsewhere. For example, has the nurse been adequately trained to use the device?

Conventionally a distinction is made between *active failures* and *latent conditions* (Reason, 2008). An example active failure would be pressing the wrong button, and hence instructing a device to deliver a drug dose ten times too high, let's say with a fatal outcome. An example of a latent condition would be failing to train the nurse to press the correct buttons. The difference is that the active failure has an immediate causal impact on the outcome, whereas the latent condition has been a potential cause for an indefinite time. Latent conditions remain concealed until a particular combination of circumstances exposes them.

There is an unbounded number of latent conditions prior to any active failure. Continuing with the example above: clearly a latent condition is that the nurse was not adequately trained to use the device, or they were not trained to double-check they have pressed the right buttons. Equally, a latent condition is that the nurse was recruited to work with the device when they did not have adequate training. Another latent condition is that the hospital procurement has acquired many different devices, thus requiring training that would not have been necessary if only one device was in use. Another is in the lax hospital procedures; two nurses should have independently checked critical clinical actions that can make a factor of ten difference in dose. Another latent condition might be the poor ergonomics of the device: buttons were too small and too close, so it was too easy to press the wrong button. Another latent condition might be the decision to have a drug container with sufficient volume to provide a fatal dose; had the device been primed with 100mL the patient could have survived even with a ten-times over-dose, because the supply would have run out. But with a 1L container, the dose was fatal. Yet another latent condition is the doctor's decision to use a drug that is fatal in high doses; there may have been alternative drugs that would have had similar clinical benefits without the risks of overdose.

Certainly the fatality occurred as an unintended side effect of the drug. This latent condition is harder to appreciate. A drug such as morphine is regularly used for pain relief. Overdoses of morphine often cause fatalities. However, it is plausible that a different molecular structure could provide pain relief without such risks. Here is a latent condition that is routinely accepted: this is what morphine *does*. Nevertheless one can imagine designing a new drug that relieves pain without other risks.

Another latent condition is that the hospital decided to buy cheaper drug delivery devices. "Smart" devices are more expensive and their higher cost cannot be justified. Now the latent condition is that a different device could have inhibited the nurse's overdose instructions, but the hospital procedures for device procurement discounted the benefits of more expensive devices. Studies show that smart devices do not measurably affect error rates because users choose to opt out of the error detection facilities (Rothschild *et al*, 2005) — itself another form of latent condition; we will return to this issue later.

It is not hard to think of more latent conditions. Because there are many latent conditions, inevitably some will be ignored. Unfortunately investigators will have a natural tendency to consider latent conditions that are specific to the circumstances. For example the nurse's training (or lack of) is a more obvious latent condition to consider than the prescription of the particular drug. After all, morphine is prescribed to many patients so there was nothing situational in this decision. In contrast, the nurse's training was a specific issue about *that* nurse on *that* day, and there is (or seems to be) a clear causal chain from their inadequate training to their unfortunate choice of actions.

Let us define *frame error* to be a latent condition that is not situational, not specific to the particular circumstances or root causes of the active failure. Thus the prescription of morphine, the pharmacy supplying it in a large container, even the chemical design of the drug, are all frame errors. Had they been otherwise, we could envisage achieving the intended outcome and avoiding the active error altogether.

Consider the following example incidents:

- (BBC, 2000) A premature baby was given a dose of 10.1 mL per hour of diamorphine instead of the intended 0.1 mL per hour, and received a toxic overdose over the 90 minutes before the incorrect dose was noticed. The responsible nurse reported that she had received no formal training on using the machine. The hospital now trains nurses how to use the device. But the incident report provides further details that make the design of the device suspect: the device was alarming every two minutes, and had to be silenced; secondly, the alarm cancel button was adjacent to the 10 button which *also* cancelled the alarm. Accidentally the nurse pressed 10 instead of cancel, and the dose changed from 0.1 to 10.1. Given that the hospital has these devices, better training is essential. But the design of the device framed the situation so that the active error was more likely.
- (ISMP, 2007b) A patient was given a fatal dose of fluorouracil 24 times too high, because in calculating an hourly dose rate for a 4-day infusion, two nurses both failed to divide by 4×24 but instead divided by 4. A root cause analysis recognized the design problems in the infusion pump, but recommended reviewing the training to ensure it covered the confusing aspects of the pump design. The report suggests introducing new design features to block similar errors, but did not suggest improving the design to avoid the errors caused by confusion.
- (Zhang, *et al*, 2003) An infusion device ignores decimal points in numbers larger than 99.9. When the nurse entered a rate of 130.1 mL per hour, the device was programmed at the rate 1,301 mL per hour — ten times too high — because the decimal point entered by the nurse was ignored by the device.

If we can avoid frame errors, because they are not situational, the benefits will be considerable.

Research chemists are searching for safer painkillers. If they succeed, then the frame of all painkilling errors will have shifted; everybody will be safer. Unfortunately, the search for safe painkillers is not certain. New painkillers may have unanticipated side-effects (including abuse: side-effects need not be "just" clinical), and the side-effects might even be delayed for years, thus reducing the ability to manage errors in their use, simply because active errors are no longer noticed because they have no immediate consequence.

What we need is a reframing that is effective. This paper proposes that the design of the medical devices is such an area. We will show that drug delivery devices, despite "working as designed," in fact induce active failures. Moreover, there are existing regulations that prohibit certain design features, and these are being ignored. The problem is that framing errors are being created by device manufacturers who are not reflecting on the consequences of their design decisions.

Devices work, hospitals buy them, nurses are trained to use them, so if errors occur, then provided the devices work as designed, the manufacturers are as far out of the frame as designing new drugs. As Johnson *et al* (2007) say most errors are attributed solely to the user and that the primary method of error prevention is to retrain the user. Of course this is a cheap option, and one might note that the device manufacturer could also provide the training.

ACTIVE AND PASSIVE DEFENSES

Reason introduced the "Swiss cheese" metaphor of error defense (Reason, 2008). Imagine a stack of holey Swiss cheese slices. An error occurs when the holes in slices are aligned and light from the hazard can shine through the stack; any slice can defend against an error by being moved so that its hole no longer aligns with the rest and the hazard cannot be seen through the stack of slices. The metaphor makes clear that errors occur for the coincidence of multiple reasons; for example, if we imagine a nurse as a defense against an error (that is, as a slice of cheese), when an error gets past the nurse the nurse is no more a unique cause, let alone to blame, than any other slice of cheese that also permitted the hazard to become an actual error.

Using the metaphor, a latent condition is a hole in a slice of cheese that was put down a long time ago, and an active failure is a hole in a slice of cheese that was positioned at (or around) the time of the error occurring.

The metaphor clearly represents *active* defenses against errors. It encourages the view that if an error occurs, in future there ought to be more slices of cheese or smaller holes in the existing slices. Thus ISMP reports (2007a, 2007b) suggest that infusion pumps should impose limits on permitted drug doses: this is a new slice of cheese, with a hole that only permits drug doses in a particular range, hopefully preventing overdoses such as the one investigated. Another suggestion is that nurses should both calculate and estimate drug doses, the estimate thus providing another slice of cheese.

The metaphor does not clearly represent *passive* defenses. If a potential error cannot occur, then the slices of cheese are irrelevant. This might be modelled by imagining the light is deflected by a mirror; for if the light is deflected, it does not matter how the holes in the slices align, whether there are more slices or not, or what size the holes are. In the ISMP (2007b) example, the pharmacy had printed all details of the drug dose on paper, but it could have printed only the correct dose for programming the infusion pump. Had it done so, the two nurses would not have needed to do any calculations at all, and the error could not have occurred. The nurses would not have needed to estimate the dose as a double check on a calculator-based result, because they would not need to calculate anything.

The most common form of passive defense is a *constraint*. For example if oxygen and nitrous oxide have incompatible connectors, they cannot be mixed up — unless the connectors are broken, or they have been mixed up by technicians installing the connectors (as has happened). In the example (BBC, 2000) above of a nurse pressing 10 instead of cancel, an obvious contraint would have been a physical cover that only permitted pressing the cancel button, and an equivalent but simpler constraint could have been the device software-disabling all buttons except stop and alarm cancel.

Another problem with the Swiss cheese model is that it does not represent ideas that are not already slices of cheese. An example would be airbags in cars; drivers have accidents that result in injury, but there is no "cheese" to represent airbags *until airbags are invented* — then, of course, once invented, they have "holes" in, just like any other defense. In user interface design, many features do not exist until they are implemented by programmers, and even then the features are abstract, only visible to experts in representations like state transition diagrams or formulae, and therefore thinking about defenses in interaction programming is unusually hard.

The Swiss cheese model also ignores thinking about *error recovery*, as it has no visual representation of what to do after an error occurs. If there is a drug overdose, say, the model cannot represent the drug antidote. In the BBC (2000) example, one could allow the nurse to press any buttons without constraint, but their audible effects should be different. Now the error can occur, but the nurse gets different feedback from the device. Pressing 10 *might* have been intentional but it should not have cancelled an alarm.

IMPROVING PASSIVE DEFENSES AND CONSTRAINTS

The ISMP (2007b) report is very detailed and provides a case study of human error resulting in a preventable error. We consider the following example latent errors in the device design:

The device (an Abbott aimplus) requires the operator to enter the actual calculated dose, such as 1.2 mL per hour, yet the nurse is provided with numbers to calculate the dose. In this case, the dose is to last over 4 days, deliver 5250 mg of fluorouracil from a container at a concentration of 45.57 mg per mL. These numbers can be used to work out the correct dose rate, 1.2 mL per hour. The nurse therefore had to perform the calculation, though it is worth noting that the device itself could have done the calculation from these numbers instead. The nurse need not have made a calculation error (though they may still have made a number entry error).

The device asks the user to review the numbers entered before an infusion can be started. Curiously the review merely consists of hitting "down arrow" repeatedly — certainly allowing the nurse to read the numbers entered, but not requiring it. It would be expected that nurses habituate or automate pressing down arrow and not actually reviewing the numbers. Also, the numbers for review are exactly the same as the numbers entered, so if a nurse erroneously enters 28.8 instead of 1.2, the review process does not help detect the error: it merely shows what the nurse (mistakenly) wanted to enter. The device could have asked the nurse to review its estimate of the duration of the infusion: this is a

number it can calculate from what it has been told, but it may have surprised the nurse to see the estimate of 4 hours rather than 4 days. The reviews would have been more reliable if they had asked the nurse to enter numbers (or approximate numbers) rather than reading — to be actively engaged in the review rather than having a chance to automate it.

Although not a reported factor in the fatality, the device does not reject number entry errors. Thus if a user enters 1.2.3 the device will act as if 1.23 has been entered, or in some modes as if 123 has been entered. In no mode is 1.2.3 (or any similar error) blocked as an error. The ISMP report envisages "smart" infusion pumps that check that numbers input are within given ranges, but it does not consider that numbers are well-formed. This is ironic because ISMP publishes a list of error prone dose designations (ISMP, 2007a) which requires that ill-formed numbers such as .5 should NEVER (its emphasis) be used. Here the reason is that .5 can be misread as 5; if 0.5 is intended, it should always be written with a leading zero. The device in the ISMP (2007) case ignores all ill-formed number errors, including errors forbidden by ISMP.

The ISMP regulations also require that micrograms are never written as μg, because the symbol μ can be misread as *m*, and hence μg misread as mg, which would be out by a factor of 1,000. Again the device ignores this rule, and represents micrograms as μg. It also uses / instead of per, another conflict with the ISMP guidance — a slash (/) can be misread as a digit 1 (e.g., 25 units/10 units can be misread as 25 and 110 units).

The device manufacturer is thus flouting the ISMP guidelines and flouting The Joint Commission National Patient Safety Goal for accredited organizations. This is clearly a latent condition. Indeed, we are unaware of any interactive drug delivery system that handles numbers correctly to these standards.

Note that fixing this class of latent condition would not affect correct use of the device.

POTENTIAL BENEFITS OF CONSTRAINING WELL-FORMED NUMBERS

Number entry systems do not constrain the user to entering only well-formed numbers. Since imposing constraints require a negligible amount of programming, it begs the question whether the benefits of the constraints are outweighed by the cost of programming to higher standards.

To help answer this question, we performed a Monte Carlo simulation of generating a valid drug dose and entering it using a numeric keypad. We assumed that the user makes keying errors with probability 0.01 (that is, on average 1 key in 100 is mispressed). Thus if the dose is 1.2, the number entered will most likely be 1.2, or with low probability it may be 0.2, 1.9, 152, etc. We then classify an adverse drug event (ADE) if the numerical value entered is more than a factor of 2 out from the intended value. Thus for an intended dose of 1.2, the numbers 0.2 and 152 are both ADEs, but 1.9 is not. The simulation showed that approximately 1.5% of numbers entered were ADEs; what was interesting was that 0.5% (i.e., a third of the ADEs) would have been blocked by well-formed number checks. For example, the intended dose of 100 might have been erroneously entered as .00, 1.., 10., all of which are clearly ill-formed numbers that should be blocked.

Note that this is not a conventional "smart" pump; it does not know what a safe range of dose is, but simply by enforcing ISMP regulations it can block a third of ADEs (as defined above) a standard pump would permit. Moreover it is not a feature that users need to opt out of under any circumstances, and therefore promises more reliable gains than conventional smart devices (Rothschild *et al*, 2005).

Vincente *et al* (2003a) estimate the number of fatalities from a particular patient-controlled analgesia (PCA) pump, the Abbott Lifecare 4100 Plus II, caused by data entry errors.[1] The Lifecare 4100 Plus is a market leader in the US PCA market and may thus be assumed to be a typical device. Scaled to the UK population (i.e., 60/300 of Vincente's numbers), the estimated mortalities are 1.73 to 11.1 per year; the range being explained by Vincente *et al*'s consideration of hospitals under-reporting and the possibility that some reports analyzed being multiple reports of the same incident (the reports are anonymous). If the Abbott Lifecare was programmed to check that numbers were well-formed (and if the Monte Carlo simulation with p=0.01 is realistic to the clinical environment) then the fatality rates would drop to 1.3 to 8.6 per year for this single device alone. Using estimates from Vincente *et al* (2003b), mortality from data entry errors on *all* PCAs ranges from 87–889, which could be reduced to 19–203, saving 68–686 lives.

This might be dismissed as a small gain, but it could be achieved simply by more careful programming, by blocking syntax errors in numeric input. British law requires that risks be reduced "as low as reasonably possible" (ALARP), which means that risks must be reduced unless the cost of doing so is disproportionate. Programming numeric entry

[1] Vicente *et al*. (2003), like most clinical literature, refer to "programming errors" meaning user errors in data entry. In the present paper, "programming errors" refers exclusively to software (or firmware) programming errors made during the manufacturing process.

properly is hardly a disproportionate cost! Moreover, routine firmware upgrades could replace the number entry code at negligible cost. Users would not need significant retraining, if any, because correct operation would be unchanged. The healthcare market of course has many infusion pumps in use, and the reduction in fatalities would be scaled up accordingly if all devices were improved to include well-formed number entry constraints.

Problems with numbers in clinical settings have been known at least as far back as 1929 (Thimbleby, 2008) and it is worrying that this knowledge (despite being explicitly formalized in the ISMP rules) has not yet been applied to the design of modern interactive devices. One is also led to wonder about other types of programming errors.

AVOIDING LATENT DESIGN CONDITIONS: UI DISCOVERY TOOLS

If improving number-entry interaction programming would save lives, why isn't it done? The most charitable explanation is that manufacturer's development processes appear to work, and therefore there is no problem to address. This happens because there is such a large gap between the development and testing of the programs and the active failures that occur with their use; moreover, the culture assumes that users will become more reliable through better training rather than by addressing the frame errors. After all, it is "just" number entry, and it seems to work.

A second reason is that manufacturers are not getting feedback from problems operating earlier versions of devices. This happens because hospitals under-report errors, because errors are treated as "user errors" rather than (potentially) latent conditions in device design, and because "near misses" are not reported.

A third reason is that testing procedures, including user-centered evaluation, are not detecting the problems. The programmer develops a number entry routine, and it seems to work. User evaluation is concerned with bigger issues, such as task analysis, ergonomic issues, mode confusion, and so on.

The real problem is that testing to cover a design is too expensive. A single three digit number entry field requires at least 12^4=20,736 (ten digits, decimal point, enter number, and four positions to press them) tests to cover all its input. No user test can last that long, and even if it did, analyzing the experiment would be tedious and thereby itself be error prone. Some test sequences would be missed, some would be performed several times. Indeed, if we grant that manufacturers are testing their devices, then they *must* have missed or ignored unlikely test cases.

Such problems with human testing are well known and easy to demonstrate (Thimbleby, 2007b), and can be addressed by using formal methods. The first hurdle to overcome is to get manufacturers to realise that testing is missing important defects, then the second hurdle is to get manufacturers to use formal methods. Formal methods generally require a level of mathematical maturity that few developers have (or even feel they need to have).

Thimbleby (2008) proposed a new drug dose calculator that detected number entry errors and constrained the user to do the correct calculation. However the calculator was developed in an entirely conventional way: the program code was just written and debugged by *ad hoc* testing. Certainly, the developer did not cover all possible test cases. In many ways, then, this program is a typical safety-critical program, and therefore a good demonstration of the value of formal methods.

A software test user was written, to perform the equivalent of a user evaluation of number entry but automatically, and thus tirelessly and providing complete coverage of all possible input, correct and erroneous. Figure 1 shows a finite state machine that was generated from this automatic process. Dark circles on the diagram represent states recognized by the device as user input errors.

The question is, are all errors detected and are no errors reported for correct input? This is a question for model checking.

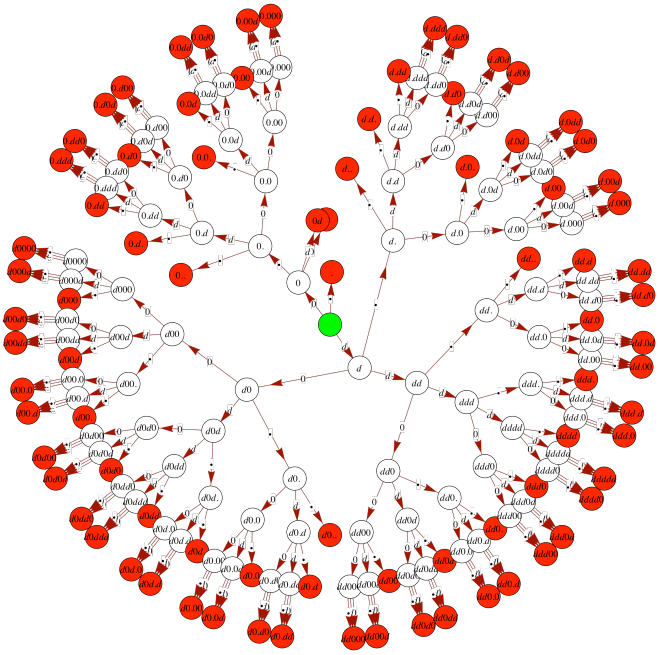

Figure 1. Automatically generated state machine for number entry. The transitions shown are for 0, decimal point, and any non-zero digit *d*. (In general each digit has its own transition, but representing this would make the diagram too complex to visualize conveniently in this paper.) The central state is the initial state, and dark states are states identified by the device as erroneous, such as those representing numbers starting with a decimal point. All states at the periphery are errors, because they represent numbers with too many digits. As reproduced in this paper, the diagram is too small to see details; instead it would normally be explored using an interactive visualization tool that permits zooming and scrolling. Model checking then establishes that all erroneous states are dark, and no dark state is not erroneous.

In conventional formal methods, model checking would be done in a specialist tool such as Alloy, SPIN or SMV, and the model checking questions would be expressed in a modal logic. This is not trivial, even though one might hope that a manufacturer would have some people capable of using model checking tools. Furthermore, using a model checker requires having in place a formal translation process between the program being checked and the model checking tool; for example, the program might be developed from a specification written in Promela. This raises the question of how this is done reliably, and what happens when user testing (iterative design) forces changes on the program — the whole, costly process has to be repeated.

Instead, we wrote a special purpose model checker in the same programming language as the calculator itself was implemented in. Thus no new notation had to be learned, no model checker/programming language translation was required, and the model checking questions were expressed in conventional programming notation, without requiring any logical connectives or notation. To make the approach more reliable, a different style of programming was used: the calculator was written in a conventional imperative fashion, but the model checking was written using regular expressions. By choosing a very different style of programming, we reduced the chances of making the same mistakes twice.

We added a new button to the calculator to generate and test models. The calculator passed the model checking tests in all number entry fields. We are therefore very confident that it works as claimed.

Further examples, based on reverse engineering of a current infusion pump are provided in Oladimeji (2009). We call the approach a User Interface (UI) Discovery Tool, for it discovers precisely what the user interface is and then checks it (Thimbleby, 2009). As pointed out in Thimbleby (2009) UI discovery brings with it, in addition to the benefits mentioned above, all of the benefits of interaction programming (Thimbleby, 2007a), including many other aspects of design analysis than we have space to discuss in this paper. Further papers are in preparation to clearly define the programming steps to implement UI discovery.

COMBINING METHODS: THE BEST OF BOTH WORLDS

We know that user centered design processes can have a major and beneficial impact on design and use in the clinical setting (Lin *et al*, 1998). However, user testing requires humans, appropriate methods to record and analyze protocols, as well as appropriate ethical treatment of participants. Although any user testing can be and generally is insightful, thorough testing is slow and expensive, and error-prone. For many systems, exhaustive UI testing is infeasible because

there is not enough time to try all necessary test cases: indeed, except for the simplest of systems, user testing rarely covers a design fully. Systematic testing with human users is also fraught with human error, not just with the system being tested, but with the whole process, of recording and analyzing results.

UI discovery, on the other hand, raises no ethical issues, it does not demand human time for testing, and it can be used repeatedly with no extra cost as a design drifts. Once a developer has decided that certain issues or properties need analysis, tests can be repeated with ease on a range of device models or versions.

A close reading of ISMP (2007b) suggests that manufacturers do not use human factors engineering methods (or if they do, they ignore the results). Thus the skills required to achieve the beneficial outcomes that Lin *et al* (1998) gain requires at least training if not also recruitment, and the management will to permit the human factors work to impact development. In contrast, UI discovery can be done by existing programmers and implemented *within* their existing skillset. Manufacturers clearly have (or have access to) programmers with sufficient skill, since their deployed devices are certainly programmed by somebody of sufficient skill to use UI discovery.

Ideally, conventional user centered methods and user evaluation would be combined with UI discovery tool analysis, and the two approaches would mutually inform each other. For example, user evaluation would identify types of problems, and model discovery would then be set up to ensure that those problems occur nowhere.

Once a UI discovery process has been set up, both developers and human factors engineers would contribute to the battery of tests it model checks. Thus using UI discovery would promote collaboration between these two fields.

CONCLUSIONS

Active failures naturally focus our attention, and approaches such as root cause analysis aim to identify causes of errors, but they are rarely pursued in enough depth to identify all latent conditions that created the context for the active errors to occur. This paper introduced the term *frame error* to make clear that a large part of the context and causes of an error are unspoken; indeed there are an unbounded number of frame causes, and not all of them can be considered. However, fixing frame errors can have a much wider impact than fixing conventional root causes, because frames apply to all similar errors, not just the situated errors that form the focus of a root cause analysis.

We suggested that a particular frame error in the healthcare domain was number entry in drug dose delivery systems. We noted that this is a real problem causing loss of life, we showed that there are national regulations to address it, yet manufacturers are ignoring these concerns. We provided some reasons why this might be, including that conventional human factors engineering (HFE) approaches cannot reliably address the problems. We proposed UI discovery as a solution, and showed that it brings benefits at low cost. It also has political benefits in that it can encourage collaboration between the programmers and human factors experts, an area usually fraught with tension.

Discovery tools allow designers and developers to retain their current processes and programming methodologies, but adds a powerful form of analysis that is essential in safety critical areas and beneficial in all other areas. Regulatory agencies should use UI discovery to test conformance to appropriate standards; manufacturers may use it in due diligence.

ACKNOWLEDGEMENTS

This work was supported by EPSRC grant EP/F020031. The author is a Royal Society-Leverhulme Trust Senior Research Fellow and acknowledges this generous support. Ann Blandford, Michael Harrison, Peter Ladkin and Patrick Oladimeji made many helpful comments.

REFERENCES

BBC (13 July, 2000) Massive drugs overdose killed baby, http://news.bbc.co.uk/1/hi/health/832313.stm

Institute of Safe Medication Practices (2004) *ISMP Canada Safety Bulletin*, **4**(1):1.

Institute of Safe Medication Practices (2007a) *List of Error-Prone Abbreviations, Symbols and Dose Designations*, http://www.ismp.org/Tools/errorproneabbreviations.pdf

Institute of Safe Medication Practices (2007b) *Flourouracil incident root cause analysis*, http://www.cancerboard.ab.ca/NR/rdonlyres/2FB61BC4-70CA-4E58-BDE1-1E54797BA47D/0/FluorouracilIncidentMay2007.pdf See also http://www.ismp.org/Newsletters/acutecare/articles/20070920.asp

Johnson, T.R., Tang, X., Graham, M.J., Brixey, J., Turley, J.P., Zhang, J., Keselman, A., and Patel, V.L. (2007) Attitudes toward medical device use errors and the prevention of adverse events, *The Joint Commission Journal on Quality and Patient Safety*, **33**(11):689–694.

Lin, L., Isla, R., Doniz, K., Harkness, H., Vincente, K.J. and Doyle, D.J. (1998) Applying human factors to the design of medical equipment: Patient-controlled analgesia, *Journal of Clinical Monitoring and Computing*, **14**:253–263.

Oladimeji, P. (2009) *Devices, Errors and Improving Interaction Design—A case study using an Infusion Pump*, Swansea University MRes Thesis.

Reason, J. (2008) Human error: models and management, *British Medical Journal*, **320**:768–770.

Rothschild, J.M., Keohane, C.A., Cook, E.F., Orav, E.J., Burdick, E., Thompson, S., Hayes, J., Bates, D.W. (2005) A controlled trial of smart infusion pumps to improve medication safety in critically ill patients, *Critical Care Medicine*, **33**(3):533–540.

Thimbleby, H. (2007a) *Press On: Principles of Interaction Programming*. MIT Press, Boston, Mass. USA.

Thimbleby, H. (2007b) User-centered methods are insufficient for safety critical systems, *Proceedings USAB'07—Usability & HCI for Medicine and Health Care*, edited by Holzinger, A., Springer Lecture Notes in Computer Science, **4799**, pp1–20, 2007.

Thimbleby, H. (2008) Ignorance of interaction programming is killing people, *ACM Interactions*, 52–57, September+October.

Thimbleby, H. (2009) Interaction programming: next steps, *Proceedings ACM CHI*, pp3811–3816.

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

Vicente, K.J., Kada-Bekhaled, K., Hillel, G., Cassano, A., Orser, B.A. (2003b) Programming errors from patient-controlled analgesia: Reply, Canadian Journal of Anesthesiology, **50**(4):856–857.

Zhang, J., Johnson, T.R., Patel, V.L., Paige, D.L., and Kubose, T. (2003) Using usability heuristics to evaluate patient safety of medical devices, *Journal of Biomedical Informatics*, **36**(1–2):23–30.